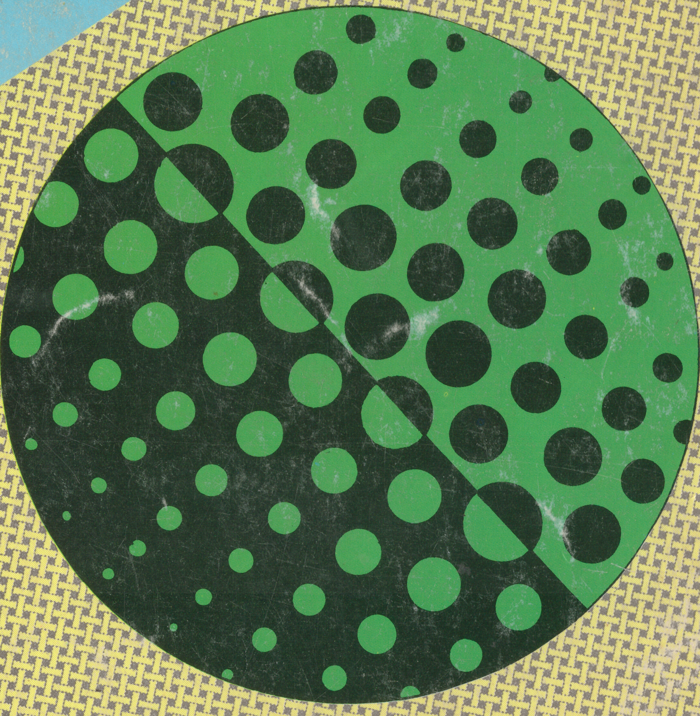
แนะนำภาษา Pascal โดย Turbo Pascal



ชัชวาล ยนต์หงส์

แนะนำภาษา Pascal
โดย
Turbo Pascal

ผู้ช่วยศาสตราจารย์ ชัชวาล ยนต์หงส์

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยสงขลานครินทร์

สำนักพิมพ์โอเดียนสโตร์

860-862 วังบูรพา กรุงเทพฯ 10200 โทร. 221-0742, 221-6567

218/10-12 สยามสแควร์ซอย 1 กรุงเทพฯ 10330 โทร. 251-4476, 254-8806 - 7

ISBN 974-276-511 - 1

แนะนำภาษา Pascal โดย Turbo Pascal

ผู้ช่วยศาสตราจารย์ ชัชวาล ยนต์หงส์

ราคา 120 บาท

พิมพ์ครั้งที่ 2 พ.ศ. 2535 จำนวน 2,000 เล่ม

พิมพ์ที่ โอ.เอส.พริ้นติ้ง เฮ้าส์
113/13 ซอยวัดสุวรรณคีรี (วัดขี้เหล็ก)
ถนนปิ่นเกล้า - นครชัยศรี เขตบางกอกน้อย
กรุงเทพมหานคร 10700 โทร. 424-6944, 424-7292
นายประสิทธิ์ สันติวัฒนา ผู้พิมพ์ ผู้โฆษณา 2535

คำนำ

หนังสือแนะนำภาษา Pascal โดย Turbo Pascal นี้ เหมาะสำหรับผู้เริ่มต้นเรียนภาษาคอมพิวเตอร์และเหมาะสำหรับที่จะใช้เป็นแบบเรียนเนื่องจาก แต่ละบทผู้เขียนพยายามอธิบายพร้อมยกตัวอย่างประกอบมากมาย ซึ่งตัวอย่างในหนังสือเล่มนี้มีมากกว่า 100 ตัวอย่าง เมื่อรวมกับแบบฝึกหัดพรอมคำเฉลย จะมีตัวอย่างทั้งสิ้นกว่า 130 ตัวอย่าง

รายละเอียดหนังสือเล่มนี้มีดังนี้คือ

1) บทที่ 1 2 3 กลาวถึง โครงสร้างหลักของภาษา คำสั่งส่งข้อมูลให้คอมพิวเตอร์และคำสั่งแสดงผลออกจอ รวมทั้งคำสั่งหลักของภาษา Pascal ซึ่ง 3 บทนี้เป็นพื้นฐานหลักที่สำคัญมาก

2) บทที่ 4 5 6 กลาวถึง ข้อมูลแบบพื้นฐานต่างๆ ได้แก่ ข้อมูลแบบอาร์เรย์ บูลีน และเรคอร์ด การศึกษาข้อมูลเหล่านี้จำเป็นและมีประโยชน์ในการประหยัดการใช้ตัวแปร โดยเฉพาะอย่างยิ่งข้อมูลแบบอาร์เรย์เป็นข้อมูลที่มีประโยชน์และสะดวกในการใช้งาน

3) บทที่ 7 กลาวถึง การสร้างโพรซิเจอร์และฟังก์ชันซึ่งทำให้การตรวจโปรแกรมง่ายและสะดวกขึ้น เนื่องจากเป็นการตัดโปรแกรมเป็นส่วนๆ

4) บทที่ 8 กลาวถึง การจัดการไฟล์ เป็นการอ่านข้อมูลจากดิสก์และบันทึกข้อมูลลงดิสก์ ซึ่งอาจไม่จำเป็นสำหรับผู้เริ่มต้นเท่าใดนัก และไม่ต่อเนื่องกับบทต่อๆไป

5) บทที่ 9 กลาวถึง ฟังก์ชันและโพรซิเจอร์มาตรฐาน กล่าวคือ คำสั่งต่างๆที่มีอยู่ใน Turbo Pascal

6) บทที่ 10 กลาวถึง โครงสร้างข้อมูลและข้อมูลแบบพอยน์เตอร์ เนื้อหาของบทนี้เป็นเป็นการแก้ปัญหาที่ค่อนข้างซับซ้อนและก็ไม่ต่อเนื่องกับบทต่อไป

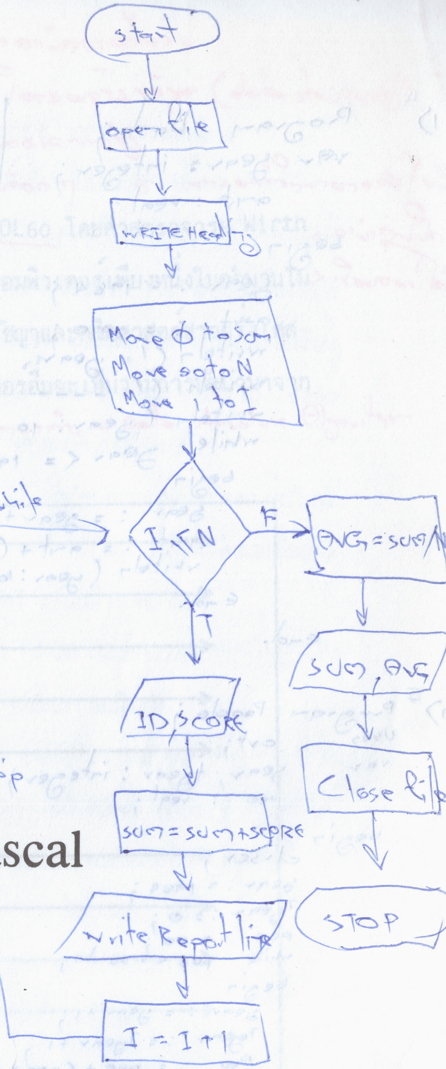
และ 7) บทที่ 11 กลาวถึง การเขียนกราฟฟิก บทนี้ผู้เขียนใช้หน้ากระดาษมากกว่าบทอื่นๆ เนื่องจาก การเขียนกราฟฟิกมีรายละเอียดมาก

เนื่องจากภาษา Pascal เป็นภาษาที่สร้างสำหรับผู้เริ่มเรียนภาษาคอมพิวเตอร์ ดังนั้นอาจกล่าวได้ว่า เมื่อศึกษาภาษา Pascal จากหนังสือเล่มนี้ จะทำให้เข้าใจถึงหลักใหญ่ๆของการเขียนโปรแกรมเพื่อการศึกษาภาษาคอมพิวเตอร์อื่นๆในอนาคต

ผู้เขียนหวังว่าหนังสือเล่มนี้คงจะมีประประโยชน์สำหรับนักเรียน นักศึกษา ตลอดจนผู้ที่ต้องการศึกษาภาษา Pascal ไม่มากก็น้อย

สารบัญ

บทที่ 1
เข้าสู่โลก Pascal



1) 4

```
Program Bank;
var year : integer;
    amt : real;
begin
    year := 1984;
    amt := 2000;
    writeln ('  year          Amount ');
    writeln ('  ----          ------ ');
    writeln ( year:10 ,  amt:10:2);
    while year <= 1999 do
      begin
        year := year+1;
        amt := amt+(amt * 0.09);
        writeln (year:10 , amt:19:2);
      End.
End.
```

2) 5

```
Program People;
Uses
    crt;
  var
    year, tyear : integer;
    peo : real;
begin
  clrscr;
  year := 1984;
  Tyear := 0;
  peo := 50;
  while peo <= 90 do;
  begin
    year := year+1;
    Tyear := Tyear+1;
    Peo := peo+(peo*0.02);
  End;
    write(' Year '   Year : 10);
    writeln ('  Tyear',  Tyear : 20);
End.
```

3) 6

```
Program Student;
uses
      crt;
var   total, upgo : integer;
      num : integer;
begin
   clrscr;
   total := 0;
   upgo :
   readln (num);
   while num<>-1 do
  begin
    total := total+1;
    if num>90 then  up90 := up90+1;
    readln (num);
  End;
  writeln('      total', total : 20 ');
  ------ ('      up90', up90 : 21 ');
End
```

4)

```
Program rcode;
uses
     crt;
var
  ycode, Fres, Sop, Jun, Sen, Gra : [illegible]
begin
  clrscr;
  Fres := 0;
  Sop := 0;
  Jun := 0;
  Sen := 0;
  Gra := 0;
  readln (ycode);
  while ycode <> 0 do
  begin
   if ycode = 1 Then Fres := Fres+
   -----------  2  ----- Sop  Sop+
   -----------  3  ----- Jun  Jun+
   -----------  4  ----- Sen  Sen+
   ----------- >4  ----- Gra  Gra+
  End.
  writeln('aaaaaaaaa');
  ------- ('Freshmen ', Fes:11);
  ------- ('Sophamore ', Sop:10
  ------- ('Junior       Jun:13
  ------- ('Senior       Sen:13
  ------- ('Graduate      Gra:11
```

5)

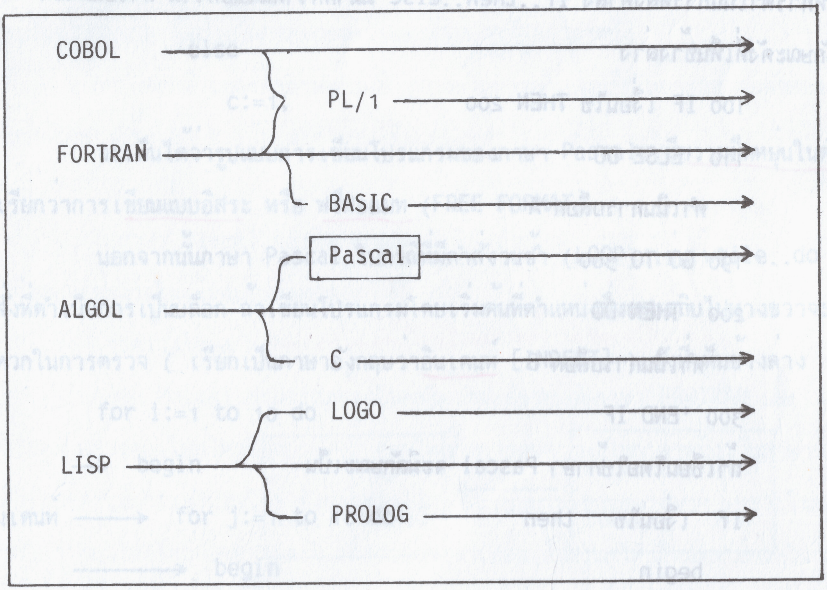
```
Program Exam;
Uses  crt;
var   Score : array[1..25] of real;
      avg, total : real;
      underavg, i : integer
begin
  clrscr;
  avg := 0;
  underavg := 0;
  total := 0;
  i := 1;
  while i <= 25 do
  begin
    write ('Please input data as time
    readln(score[i]);
    total := total + score[i];
    i := i+1;
  End;
  avg := total / 25;
  writeln('    SCORE ');
  i := 1;
  while i <= 25 do
  begin
    writeln (Score[i] : 10 : 2);
    if SCORE[i] < avg then
  underavg := underavg + 1;
    i := i+1;
  End;
  writeln('  ____________________ ');
  writeln('   AVERAGE : ', avg :
```

1.1 ภาษา Pascal คืออะไร

< ประวัติ >

ภาษา Pascal เป็นภาษาที่พัฒนามาจากภาษา ALGOL60 โดยศาสตราจารย์ Wirth จากสถาบันเทคโนโลยีแห่งซูริคที่สวิตเซอร์แลนด์ ซึ่งนับเป็นภาษาคอมพิวเตอร์เพียงหนึ่งในจำนวนไม่มากที่ไม่ได้พัฒนาโดยคนอเมริกัน ชื่อ Pascal นั้นตั้งตามชื่อนักปรัชญาและคณิตศาสตร์ชาวฝรั่งเศส

เมื่อเปรียบเทียบภาษา Pascal กับภาษาคอมพิวเตอร์อื่นจะเห็นว่ามีการพัฒนามาจากรากฐานที่แตกต่างกันดังตารางข้างล่าง



ภาษาคอมพิวเตอร์ในปัจจุบัน มีรากฐานมาจากภาษา COBOL FORTRAN ALGOL LISP ซึ่งภาษาทั้ง 4 พัฒนาในราวทศวรรษ 2500

ภาษา COBOL (COmmon Business Oriented Languge) เป็นภาษาที่ใช้งานทางด้านธุรกิจ ภาษา FORTRAN (FORmula TRANslation languge) ใช้ในการคำนวณและวิจัยทางวิทยาศาสตร์ ส่วนภาษา BASIC(Beginners All purpose Symbolic Instruction Code) นั้นพัฒนามาจากภาษา FORTRAN นับเป็นภาษาที่เหมาะสำหรับผู้เริ่มต้นศึกษาภาษาคอมพิวเตอร์โดยไมโครคอมพิวเตอร์

ภาษา Pascal และ ภาษา C เป็นภาษาที่พัฒนามาจากภาษา ALGOL ซึ่งเป็นภาษาที่เน้นโครงสร้าง แต่ตัวภาษา ALGOL เองกลับไม่เป็นที่แพร่หลายนัก

ส่วนภาษา LISP และ PROLOG เป็นภาษาที่ใช้ในการวิจัยเกี่ยวกับปัญญาประดิษฐ์

< ลักษณะของภาษา Pascal >

ภาษา Pascal เป็นภาษาที่มีโครงสร้างทันสมัย มีคำสั่งต่างๆเช่น if..then..else while...do case...of ฯลฯ และยังมีคำสั่งดำเนินการเป็นบล็อก (begin..end) ซึ่งนับเป็นลักษณะเด่นของภาษา Pascal ทำให้สามารถเขียนโปรแกรมได้อย่างสละสลวยและมีขั้นตอน เมื่อเปรียบเทียบกับภาษา BASIC จะเห็นได้ว่าภาษา BASIC นอกจากไม่มีคำสั่ง while..do และ case..of แล้ว ภาษา BASIC ยังไม่มีคำสั่งที่ดำเนินการเป็นบล็อกอีกด้วย

เมื่อเปรียบเทียบภาษา BASIC กับภาษา Pascal โดยใช้คำสั่ง if..then..else และให้การดำเนินการหลังคำสั่ง if..then..else มีมากกว่าหนึ่งข้อความ ถ้าเป็นภาษา BASIC จะมีลักษณะดังที่เห็นข้างล่าง

```
100 IF  เงื่อนไข THEN 200
110  'ELSE DO
    ดำเนินการบล็อก A
190 GO TO 300
200  'THEN DO
    ดำเนินการบล็อก B
300  'END IF
```

ถ้าเขียนโดยใช้ภาษา Pascal จะมีลักษณะเป็น

```
if   เงื่อนไข   then
  begin
     ดำเนินการบล็อก A
  end
else
  begin
     ดำเนินการบล็อก B
  end;
```

จะเห็นได้ว่า ภาษา Pascal มีโครงสร้างที่สละสลวยและเข้าใจได้ง่ายกว่า เพราะสามารถใช้คำสั่ง begin...end ในเวลาที่ต้องการดำเนินการมากกว่าหนึ่งข้อความ สำหรับภาษา BASIC ต้องใช้คำสั่ง goto ซึ่งทำให้สับสนและผิดพลาดได้ง่ายกว่าในกรณีที่เป็นโปรแกรมที่มีความยาวมาก

< โปรแกรมที่มีอิสระในการเขียน >

ภาษา Pascal ไม่มีการกำหนดเป็นการตายตัวว่าต้องเริ่มเขียนจากที่ใด และในหนึ่งบรรทัดจะเขียนกี่ข้อความก็ได้ การแยกข้อความกับข้อความนั้น ไม่ได้อยู่ที่การขึ้นบรรทัดใหม่แต่อยู่ที่การใช้ตัว ; (เซมิโคลอน) ใส่คั่นระหว่างข้อความ ดังนั้นจึงสามารถเขียนได้ดังตัวอย่างที่เห็น

```
     if a=0 then b:=1 else c:=1;
หรือ  if a=0 then
          b:=1
       else
          c:=1;
```

จะเห็นได้ว่ารูปแบบการเขียนโปรแกรมของภาษา Pascal มีความยืดหยุ่นในตัวเอง ซึ่งเรียกว่าการเขียนแบบอิสระ หรือ ฟรีฟอแมท (FREE FORMAT)

นอกจากนั้นภาษา Pascal ในกรณีที่มีคำสั่งวนซ้ำ (LOOP) เช่น while..do หรือคำสั่งที่ดำเนินการเป็นบล็อก ถ้าเขียนโปรแกรมโดยเริ่มต้นที่ตำแหน่งซึ่งกระเถิบไปทางขวาจะทำให้สะดวกในการตรวจ (เรียกเป็นภาษาอังกฤษว่าอินเดนท์ [INDENT]) ดังที่เห็นข้างล่าง

```
         for i:=1 to 10 do
     -->    begin
อินเดนท์ ---->  for j:=1 to 10 do
     ------>      begin
                     ข้อความ
                  end
            end;
```

< การประกาศตัวแปร >

ในภาษา Pascal ต้องมีการประกาศตัวแปรทุกตัวแปรที่ใช้ในโปรแกรม โดยประกาศที่ตอนบนของโปรแกรมด้วยคำสั่ง var ดังตัวอย่าง

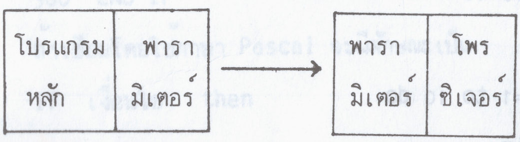
```
var a,b,c:integer;
    x,y:real
```

เมื่อประกาศตัวแปรดังตัวอย่างแล้วเราก็สามารถนำเอาตัวแปร a,b และ c มาเป็นตัวเลขจำนวนเต็ม และใช้ตัวแปร x และ y เป็นเลขจำนวนจริง

< โพรซิเจอร์และพารามิเตอร์ >

ในภาษา Pascal จะมีการแบ่งโปรแกรมออกเป็นส่วนๆ โดยปรกติจะประกอบไปด้วยโปรแกรมหลักและโปรแกรมย่อย(โพรซิเจอร์) ในการผ่านค่าจากโปรแกรมหลักไปยังโพรซิเจอร์จะใช้ตัวผ่านค่าซึ่งเรียกว่าพารามิเตอร์ ซึ่งมีลักษณะการดำเนินการดังที่เห็นข้างล่าง

```
procedure abc(x,y:real);
   var i,j:integer;  (ตัวแปรที่ใช้ได้เฉพาะในโพรซิเจอร์ abc)
     begin
       .
       .                 ผ่านค่า
       .
     end;
begin
   abc(10.0,20.0);  (เรียกโพรซิเจอร์ abc)
end.
```



< การเรียกตัวเองของโพรซิเจอร์ >

ภาษา BASIC,FORTRAN โพรซิเจอร์ไม่สามารถดำเนินการเรียกตัวเอง แต่ในภาษา Pascal มีคำสั่งดังกลาว ซึ่งเป็นประโยชน์ในการเขียนโปรแกรมเก็บข้อมูลที่มีลักษณะเป็นทรี(TREE) การเขียนกราฟฟิกแบบเรียกตัวเอง (RECURSIVE GRAPHICS) และในการแก้ปัญหาที่ต้องใช้การเรียกตัวเอง

< Pascal กับ C >

ทั้ง Pascal และ C ต่างก็พัฒนามาจากภาษา ALGOL ซึ่งมีความคล้ายคลึงกันทางด้านโครงสร้างเป็นอย่างมาก ข้อแตกต่างระหว่างภาษา Pascal กับ C ภาษา Pascal เหมาะสำหรับผู้เริ่มต้นศึกษาภาษาคอมพิวเตอร์ ส่วนภาษา C นั้นเหมาะที่จะนำไปใช้งาน

< ข้อมูลแบบต่างๆ >

ในภาษา BASIC และ FORTRAN มีข้อมูลแบบเลขจำนวนจริง แบบเลขจำนวนเต็มและอาร์เรย์ แต่ในภาษา Pascal และ C มีแบบข้อมูลที่ซับซ้อนคือ ข้อมูลแบบเรคอร์ด และข้อมูลแบบพอยน์เตอร์ ซึ่งเป็นพื้นฐานและเป็นสิ่งจำเป็นสำหรับการสร้างข้อมูลที่มีโครงสร้างแบบลิสท์ (list) และแบบทรี (tree) นอกจากนั้นยังมีข้อมูลแบบที่ผู้ใช้สร้างเอง แบบกำหนดขอบเขต และแบบเซทซึ่งนับเป็นภาษาคอมพิวเตอร์ที่มีแบบข้อมูลมากที่สุด

การเขียนโปรแกรมที่ดีต้องใช้แบบข้อมูลที่เหมาะสมและมีอัลกอริทึมที่ดี อัลกอริทึม ก็คือกระบวนแก้ปัญหาอย่างเหมาะสม ตามลำดับขั้นตอนที่ได้วางแผนไว้ล่วงหน้า ภาษา Pascal มีแบบข้อมูลเป็นจำนวนมาก จึงนับเป็นภาษาคอมพิวเตอร์ที่สมบูรณ์ที่สุดภาษาหนึ่ง

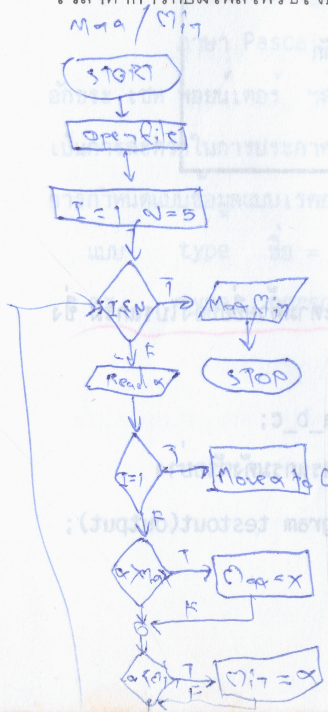
ภาษาคอมพิวเตอร์ทุกภาษาต้องมีกฎเกณฑ์ในการคำนวณหรือให้ค่าระหว่างตัวแปรที่แบบต่าง ยกตัวอย่างเช่น ถ้า k เป็นเลขจำนวนเต็มและให้ค่า 1.0 ดังที่เห็น

k = 1.0

ในภาษา Pascal จะเกิด error เนื่องจากภาษา Pascal เป็นภาษาที่มีแบบข้อมูลมาก จึงต้องเข้มงวดในเรื่องแบบตัวแปร

< การสร้างโมดูล >

ภาษา Pascal สามารถสร้างโมดูลได้เช่นเดียวกันกับภาษา FORTRAN และ C การสร้างโมดูลคือการคอมไพล์โพรซิเจอร์ (คอมไพล์คือการแปลภาษาใดภาษาหนึ่งเป็นภาษาที่เครื่องคอมพิวเตอร์ดำเนินการคำนวณได้) ซึ่งการสร้างโมดูลจะเป็นประโยชน์อย่างมาก เพราะไม่ต้องเสียเวลาทำการคอมไพล์โพรซิเจอร์ทุกครั้ง และช่วยแบ่งงานออกเป็นส่วนๆให้กับคนหลายคนทำงานได้



1.2 โครงสร้างโปรแกรมภาษา Pascal

ภาษา Pascal มีโครงสร้างตามรูปที่เห็นอยู่ โดยมีการกำหนดแบบต่างๆ หลังจากนั้นจึงเริ่มเขียนโปรแกรมหลัก การกำหนดแต่ละส่วนจะใช้คำสั่งต่างๆดังต่อไปนี้

1	program	ชื่อโปรแกรม
2	label	กำหนดชื่อลาเบล
3	const	กำหนดตัวคงที่
4	type	กำหนดแบบข้อมูล
5	var	ประกาศตัวแปร
6	procedure function	ส่วนโปรแกรมย่อย หรือ ฟังก์ชั่น
7	begin ---- end.	โปรแกรมหลัก

1 ชื่อโปรแกรม

ส่วนแรกของโปรแกรมนี้ เริ่มจากคำว่า program และตามด้วยชื่อของโปรแกรม ซึ่งสามารถตั้งชื่ออะไรก็ได้ดังตัวอย่าง

ex program test; program a_b_c;

สำหรับใน Standard Pascal ต้องมีวงเล็บหลังชื่อโปรแกรมดังตัวอย่าง

program demo(input,output); หรือ program testout(output);

2 กำหนดชื่อ label

โปรแกรมส่วนนี้เป็นส่วนที่กำหนดไว้สำหรับในกรณีที่ในคำสั่ง goto (เป็นคำสั่งเมื่อต้องการให้การดำเนินการกระโดดไปที่ตำแหน่งอื่น) โดยปรกติในภาษา Pascal คำสั่ง goto ไม่ได้เป็นคำสั่งที่ใช้มากนัก ดังนั้น คำสั่งกำหนดชื่อ label จึงไม่คอยได้ใช้เช่นเดียวกัน ชื่อของ label อาจตั้งเป็นชื่อที่ภาษาอังกฤษหรืออาจจะใช้ตัวเลขก็ได้ แต่สำหรับ Standard Pascal ใช้ได้เฉพาะตัวเลขในกรณีที่ใช้ label มากกว่าหนึ่งตัวก็สามารถใช้คอมมา (,) คั่นและเขียนชื่อ label ต่อไป

แบบ label ชื่อ,ชื่อ - -;

ex label exit,start;

3 กำหนดตัวคงที่

การกำหนดตัวคงที่นั้น const เป็นคำสั่งที่ใช้ในการกำหนดส่วนนี้ของโปรแกรม การใช้คำสั่ง const ใช้ในกรณีที่มีการใช้ค่าคงที่บ่อยๆ เมื่อกำหนดค่าคงที่แล้วก็สามารถใช้ตัวคงที่นั้นได้ทุกเวลาที่ต้องการ แต่ในกรณีที่ไม่ใช้ตัวคงที่ก็ไม่จำเป็นต้องเขียนส่วนนี้ สำหรับการกำหนดตัวคงที่กำหนดได้ดังตัวอย่าง

แบบ const ชื่อ = ตัวเลข;

```
ex    const n=28.0;
            pai=3.1415927;
```

4 กำหนดแบบข้อมูล

ภาษา Pascal มีแบบของข้อมูลอยู่มากมายตั้งแต่ เลขจำนวนจริง จำนวนเต็ม ตัวอักขระ เซท พอยน์เตอร์ ฯลฯ ถ้าหากสามารถกำหนดแบบและขอบเขตของข้อมูลล่วงหน้าได้ ก็จะเป็นการสะดวกในการประกาศตัวแปรและการตรวจโปรแกรม ตัวอย่างที่เห็นข้างล่างเป็นตัวอย่างการกำหนดแบบข้อมูลแบบเรคอร์ด ซึ่งจะอธิบายถึงรายละเอียดในภายหลัง

แบบ type ชื่อ = แบบ;

```
ex    type  person = record
                     name : string[20];
                     year : integer;
                     address : string[20]
                 end;
```

5 ประกาศตัวแปร

ภาษา Pascal จะต้องมีการประกาศตัวแปรทุกตัวแปรก่อนที่จะนำตัวแปรเหล่านั้นไปใช้หรือก่อนที่จะเขียนโปรแกรมหลัก การประกาศตัวแปรจะเริ่มจากการใช้คำสั่ง var ตามด้วยตัวแปรและแบบข้อมูล ในกรณีที่มีตัวแปรแบบเดียวกันอยู่มากกว่าหนึ่ง ให้ใช้ (,) คั่น และเขียนตัวแปรต่อไป ดังตัวอย่าง

แบบ var ชื่อตัวแปร,ชื่อตัวแปร --- : แบบข้อมูล;

ex var a,sum:integer;

b:real;

6 ฟังก์ชันและโพรซิเจอร์

ฟังก์ชันและโพรซิเจอร์มีลักษณะคล้ายคำสั่งที่ผู้เขียนโปรแกรมสร้างขึ้นเอง เวลาที่เขียนโปรแกรมที่มีความยาว การใช้ฟังก์ชันและโพรซิเจอร์จะทำให้ตรวจโปรแกรมได้ง่าย ซึ่งรายละเอียดจะอธิบายในภายหลัง

7 โปรแกรมหลัก

ส่วนสุดท้ายของโปรแกรมนี้เป็นส่วนที่ใช้ในการดำเนินการโดยตรง โดยมี begin และ end. ครอบหัวและท้ายตามลำดับ ดังตัวอย่าง สำหรับความหมายของโปรแกรมนี้ ถ้าหากผู้อ่านศึกษาหนังสือเล่มนี้ก็สามารถเข้าใจได้ ในที่นี้ต้องการยกมาเป็นเพียงตัวอย่างให้เห็นโครงสร้างเท่านั้น

ตัวอย่างที่ 1.1 โปรแกรมตัวอย่างแสดงโครงสร้างทั่วไปของโปรแกรม

```
program space;
var i:integer;
   procedure spc(n:integer);
      var i:integer;
      begin
         for i:=1 to n do
         write(' ')
      end;
begin
   for i:=1 to 10 do
      begin
         spc(i);
         writeln('Hello Pascal !')
      end
end.
```

1.3 ชื่อและแบบของข้อมูล

< ชื่อ >

การกำหนดชื่อของตัวแปร ตัวคงที่ แบบข้อมูล โพรซิเจอร์ และโปรแกรมนั้น ชื่อจะต้องอยู่ภายในกฎดังต่อไปนี้

เริ่มด้วยตัวอักษรอังกฤษและอาจตามด้วยตัวเลข

ตัวอักษรใหญ่กับเล็กถือว่าเหมือนกัน เช่น a_3 กับ A_3 ถือว่าเป็นชื่อเดียวกัน

ความยาวของชื่อจะให้ยาวเท่าใดก็ได้ แต่ใน Standard Pascal ความยาวของชื่อถูกกำหนดไว้ไม่ให้เกิน 8 ตัว

ห้ามใช้คำสงวน เช่น begin end if เป็นต้น

คำมาตรฐานสามารถใช้เป็นชื่อได้โดยไม่เกิด error เพียงแต่จะทำให้สับสน ดังนั้นควรหลีกเลี่ยงการใช้คำเหล่านี้

อันเดอร์บาร์ (_) ถือเป็นตัวอักษรอังกฤษ

ex

ถูก	ผิด
a1	1a (เริ่มจากตัวเลข)
a_b	a! (ใช้อักขระที่ไม่ใช่ตัวอักษรและตัวเลข)
	for (ใช้คำสงวน)
	a b (มีช่องว่าง)

< คำสงวนและคำมาตรฐาน >

คำสงวนก็คือ คำสั่งประเภท begin,end,if เป็นต้น คำสั่งเหล่านี้ถูกกำหนดขึ้นมาในภาษา Pascal เมื่อใช้คำสั่งเหล่านี้ คอมพิวเตอร์จะคอมไพล์ตามที่ได้กำหนดไว้

ส่วนคำมาตรฐานคือ คำสั่งประเภท write,abs,sin,cos เป็นต้น คำสั่งเหล่านี้เป็นคำสั่งประเภทฟังก์ชันและโพรซิเจอร์ที่สร้างขึ้นมา

โดยปรกติแล้วคำสงวนที่มีอยู่ในคอมไพเลอร์ทุกชนิดจะมีลักษณะเหมือนกันเกือบหมด แต่คำมาตรฐานจะแตกต่างกันตามชนิดของคอมไพเลอร์

คำสงวนกับคำมาตรฐานมีดังนี้

absolute	file	mod	shr
and	for	nil	string
array	forward	not	then
begin	function	of	to
case	goto	or	type
const	if	packed	unit
div	implementation	procedrue	until
do	in	program	uses
downto	inline	record	var
else	interface	repeat	while
end	interrupt	set	with
external	label	shl	xor

คำสงวน

abs	exit	memavail	rmdir	
addr	exp	mkdir	round	
append	filepos	move	seek	
arctan	filesize	new	seekeof	
assign	fillchar	odd	seekeoln	
blockread	flush	ofs	seg	
blockwrite	frac	ord	settextbuf	
chdir	freemem	paramcount	sin	
chr	getdir	paramstr	sizeof	
close	getmem	pi	sptr	
concat	halt	pos	sqr	
copy	hi	pred	sqrt	
cos	inc	ptr	sseg	
cseg	insert	random	str	
dec	int	randomize	succ	
delete	ioresult	read	swap	
dispose	length	readln	trunc	
dseg	ln	release	truncate	
eof	lo	rename	upcase	
eoln	mark	reset	val	
erase	maxavail	rewrite	write	writeln

คำมาตรฐาน (คำมาตรฐานสำหรับกราฟฟิกอยู่บทที่ 11)

< แบบข้อมูลพื้นฐาน >

ภาษา Pascal มีแบบของข้อมูลมากกว่าภาษาอื่น และเข้มงวดในเรื่องการตรวจแบบของข้อมูลมาก ในที่นี้จะกล่าวถึงแบบของข้อมูลที่เป็นพื้นฐานในการเขียนโปรแกรมโดยข้อมูลที่จะกล่าวถึงเป็นพื้นฐานในการทำความเข้าใจโปรแกรมในบทต่อๆ

1) ข้อมูลแบบเลขจำนวนเต็ม (integer)

-32768 <= integer <= 32767

ขอบเขตของข้อมูลแบบนี้อยู่ระหว่าง -32678 ถึง 32767 สำหรับวิธีการประกาศตัวแปรนั้นสามารถเขียนได้ดังนี้

```
var i,j:integer;
```

เมื่อประกาศตัวแปรเรียบร้อยแล้ว ก็สามารถใช้ i,j เป็นตัวแปรแบบเลขจำนวนเต็ม

2) ข้อมูลแบบเลขจำนวนจริง (real)

$$-10^{-38} <= real <= +10^{38}$$

ข้อมูลแบบนี้เป็นข้อมูลตัวเลขที่มีทศนิยม สำหรับขอบเขตของข้อมูลแบบนี้มีค่าอยู่ตามที่เห็น มีความละเอียดในการแสดงค่า 11 หลัก ยกตัวอย่างเช่น 0.5 หรืออาจเขียนเป็น 5.0E-01 ก็ได้ 5.0E-01 ก็คือ 5 คูณ 10 ยกกำลัง -1 (5.0*0.1) ซึ่งเป็นการแสดงข้อมูลแบบนี้ (ดูบทที่ 2)

วิธีประกาศตัวแปรสามารถเขียนได้ดังนี้ คือ

```
var x,y:integer;
```

3) ข้อมูลแบบอักขระ (character)

ข้อมูลแบบนี้เป็นข้อมูลที่เก็บตัวแปรเป็นตัวอักขระต่างๆ ตัวอักขระดังกล่าวนี้ถ้าหากว่าเขียนทีละตัวจะต้องมีเครื่องหมาย ( ) อยู่หน้าหลังตัวอักขระนั้น การประกาศตัวแปรสามารถเขียนได้ดังต่อไปนี้

```
     var c:char;

ex   var x,y:real;
         a:char;
     begin
       x:=1.56; y:=1.5E+5; a:='*';
```

1.4 ข้อความ

< ข้อความและเครื่องหมายคั่นข้อความ >

ลองเริ่มจากข้อความที่เข้าใจง่ายที่สุดก่อน นั่นคือการให้ค่าแก่ตัวแปรซึ่งจะใช้ := คือเครื่องหมายแสดงการให้ค่า a ดังที่เห็น a:=10

ความหมายของ a:=10 ก็คือ ให้ตัวแปร a มีค่าเป็น 10 การให้ค่าแก่ตัวแปรนอกจาก = (เครื่องหมายเท่ากับ) แล้วยังต้องมี : (โคลอน) อีกตัวหนึ่งด้วย

ข้อความในภาษา Pascal นั้น สามารถเขียนหลายๆข้อความได้ในหนึ่งบรรทัด ดังนั้นจึงจำเป็นต้องมีเครื่องหมาย ; (เซมิโคลอน) เพื่อเป็นสัญลักษณ์ในการแยกระหว่างข้อความหนึ่งการเขียนโปรแกรมที่เห็นอยู่ข้างล่างทั้งสองแบบมีความหมายเหมือนกัน

```
แบบที่ 1) if a=0 then b:=b+1 else c:=c+1; a:=a+1;
แบบที่ 2) if a=0 then b:=b+1
                    else c:=c+1;        ; คั่นระหว่างข้อความ
               a:=a+1;
```

ความหมายคือ ถ้า a=0 ให้เพิ่มค่าเก่าของ b เข้าไปอีก 1 และเพิ่มค่าเก่า a อีก 1 ด้วย ถ้า a ไม่เท่ากับ 0 ให้เพิ่มค่า c จากของเก่าเข้าไป 1 และเพิ่มค่าเก่า a อีก 1

จากตัวอย่างทั้ง 2 แบบ จะเห็นได้ว่า if..then..else ถือเป็นเพียงหนึ่งข้อความ ดังนั้น จึงใช้ ; หลัง c:=c+1 เพียงตัวเดียว ส่วน a:=a+1 ก็ถือเป็นหนึ่งข้อความเช่นกัน

< บล็อก >

ข้อความที่ใช้ในการคำนวณ แทนค่า หรืออื่นๆเช่น คำสั่งอินพุทเอาท์พุท ฯลฯ ถือว่าเป็นข้อความหนึ่งข้อความ แต่ถ้าหากว่าข้อความต่างๆเหล่านี้มารวมกันมากๆจะเรียกว่า บล็อก ในภาษา Pascal การดำเนินการทั้งบล็อกจะใช้ begin..end ครอบบล็อกดังกล่าว ซึ่งสามารถใช้กับคำสั่ง if..then..else หรือคำสั่งวนซ้ำเช่น while..do หรือ for..to..do ดังตัวอย่าง

```
if a=0 then
   begin
      ข้อความมากกว่า 1 ข้อความ
   end;
```

< คอมเมนท์ >

ภาษา Pascal ในกรณีที่ต้องการที่จะเขียนคำอธิบายหรือหมายเหตุ สามารถใช้วงเล็บปีกกา { และ } เขียนปิดหัวท้ายส่วนนั้น ซึ่งเรียกว่า คอมเมนท์ (comment) ดังตัวอย่าง

```
{ example }
{ this program is written by mr X }
```

การใช้คอมเมนท์ ไม่ทำให้เวลาการคอมไพล์ช้าลง ดังนั้น จึงควรเขียนคอมเมนท์ในกรณีที่จำเป็นเพื่อที่จะเป็นการอธิบายโปรแกรมไปในตัวด้วย นอกจากที่จะใช้ { กับ } แล้ว (* กับ *) ก็มีคุณสมบัติการดำเนินการเหมือนกัน

< อินเดนท์ >

การเขียนโปรแกรม Pascal นั้น ในกรณีที่ใช้คำสั่งวนซ้ำ (loop) หรือคำสั่งบล็อกแล้ว เวลาเขียนโปรแกรม ถ้ากระเถิบคำสั่งภายในบล็อกไปทางขวา จะทำให้สามารถเข้าใจโปรแกรมได้ง่าย การกระเถิบดังกล่าวเรียกว่า อินเดนท์ (indent) ซึ่งการใช้อินเดนท์นั้นไม่มีหลักเกณฑ์ตายตัว ในหนังสือเล่มนี้จะใช้วิธีดังที่เห็นข้างล่าง

```
program
var .....
→ procedure
──→ begin
        ┌──────────┐
────→   │ ข้อความ   │
        └──────────┘
──→ end;
begin
→ for i:=1 to 10 do
──→ begin
        ┌──────────┐
─────→  │ ข้อความ   │
        └──────────┘
    end
end.
```

1.5 โอเปอร์เรเตอร์

ภาษา Pascal มีตัวโอเปอร์เรเตอร์ที่ใช้ในการคำนวณดังตารางข้างล่าง นอกจากนี้ยังมีโอเปอร์เรเตอร์ที่ใช้ในการเปรียบเทียบ และโอเปอร์เรเตอร์ที่ใช้ทางคณิตศาสตร์บูลีนอีกด้วย

การคำนวณ	เลขจำนวนเต็ม	เลขจำนวนจริง
บวก	+	+
ลบ	-	-
คูณ	*	*
หาร	div	/
เศษ	mod	ไม่มี

จะสังเกตเห็นได้ว่า โอเปอร์เรเตอร์ที่ใช้ในการคำนวณของเลขจำนวนเต็มกับจำนวนจริงมีจุดที่แตกต่างกัน ในการหารเลขจำนวนเต็มใช้ div แต่การหารเลขจำนวนจริงใช้ / การใช้ mod และ div จะต้องเขียนเลขหรือตัวประกอบให้ห่างกับ mod หรือ div หนึ่งช่อง

นอกจากนี้ ในภาษา Pascal ยังสามารถคำนวณผสมระหว่างจำนวนจริงกับจำนวนเต็มได้อีกด้วย และผลที่ได้จะเป็นเลขจำนวนจริง ยกตัวอย่างเช่น ถ้า i เป็นเลขจำนวนเต็มเมื่อคำนวณ i*3.14 ผลที่ได้จะเป็นเลขจำนวนจริง

ส่วนการแทนค่าจากเลขจำนวนเต็มให้กับเลขจำนวนจริงได้ แต่ในทางกลับกันทำไม่ได้

ในกรณีที่มีโอเปอร์เรเตอร์หลายตัว จะดำเนินการจากคำสั่ง div mod * / + - ตามลำดับดังตัวอย่าง 20/5*4-32 div 5*4 +18 mod 4

ผลที่ได้คือ (20/5)*4-(32 div 5)*4+(18 mod 4) = 16 -24+ 2 = -6

ex

```
var a,b:integer;
    c,d:real;
begin
  a:=50 div 4;        ให้ a มีค่า 50 หาร 4 (12)
  b:=50 mod 4;        ให้ b มีค่าเศษของ 50 หาร 4 (2)
  c:=50/4;            ให้ c มีค่า 50 หาร 4 (12.5)
  d:=a*3.14;          ให้ d มีค่า 12*3.14 (37.68)
```

Coffee Break

< การกำหนดคาคงที่ตามชื่อแบบขอมูล >

Turbo Pascal มีการกำหนดแบบค่าคงที่ตามชื่อแบบข้อมูล โดยใช้คำสั่ง const มีหลักในการกำหนดดังนี้คือ

```
const ตัวคงที่ : แบบขอมูล = ( ' คาคงที่ ',
                              ' .. ',
                              ' คาคงที่ ');
```

ตัวอย่างที่เห็นข้างล่างนี้เป็นตัวอย่าง การใช้ค่า const car[1] ถึง car[6]

```
program car_race;
uses crt;
type car_array=array[1..6] of string[6];
const car:car_array=('  ()  ',
                     'O-UU-O',
                     '  UU  ',
                     'O-UU-O',
                     '      ',
                     '      ');
var k,tb:integer;
begin
  repeat
    tb:=random(50);
    for k:=1 to 6 do
      writeln('I',' ':tb,car[k],'I':50-tb);
  until keypressed
end.
```

บทที่ 2

คำสั่งอินพุทเอาท์พุท

2.1 สตาร์ท Turbo Pascal

ในบทนี้จะกล่าวถึงขั้นตอนการทำงาน และวิธีเลือกเมนูของ Turbo Pascal เพื่อทำการเขียน แก้ไขที่ผิด และดำเนินการโปรแกรม

```
A>TURBO
 File    Edit    Run    Compil    Options    Debug    Break/watch
 ===================================Edit=============================
     Line 1    Col 1    Insert Indent         Unindent    A:NONAME.PAS

                          Turbo Pascal

                          Version 5.0
                     Copyright (c) 1983, 1988 by
                     Borland International, Inc.
```

เมื่อกดปุ่มใดๆ ที่เมนู FILE จะสว่าง และใช้เคอร์เซอร์คีย์ (←↕→) เลื่อนไปที่เมนูและกด [enter] เพื่อเลือกคำสั่งที่ต้องการ ในกรณีที่ต้องการยกเลิกคำสั่งปัจจุบันให้กด Esc การใช้งานเมนูเบื้องต้นมีดังนี้

F (File) และ L (Load) เพื่อเขียนชื่อโปรแกรมที่ต้องการเขียน แก้ไข หรือดำเนินการ

E (Edit) เพื่อเข้าสู่โปรแกรมที่ต้องการเขียนหรือแก้ไข เมื่อเขียนหรือแก้เสร็จ กด F_{10} (เมื่อกด L และเขียนชื่อไฟล์แล้ว ไม่จำเป็นต้องกด E)

C (Compile) เพื่อให้เครื่องคอมไพล์ภาษา Pascal เป็นภาษาที่เครื่องทำงานได้ ในกรณีที่เกิด ERROR เคอร์เซอร์จะปรากฏตรงตำแหน่งที่เกิด ERROR

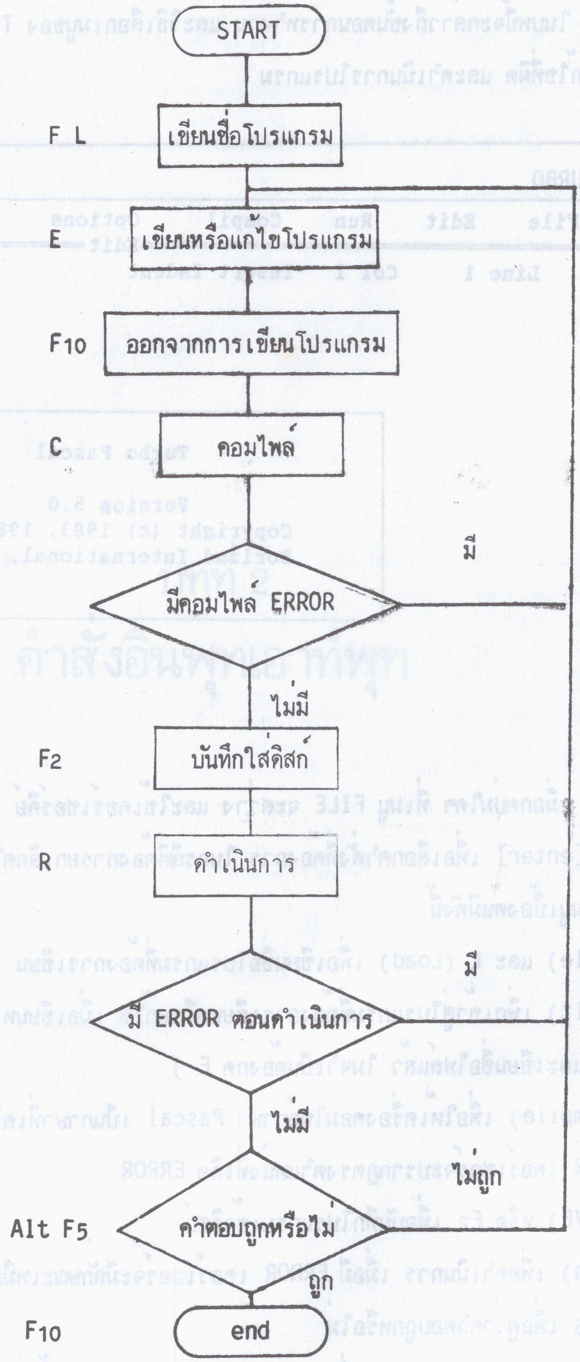
S (SAVE) หรือ F_2 เพื่อบันทึกโปรแกรมเข้าดิสก์

R (Run) เพื่อดำเนินการ เมื่อมี ERROR เคอร์เซอร์จะมีลักษณะเหมือนตอนกด C

Alt F_5 เพื่อดูว่าคำตอบถูกหรือไม่

F (File) และ Q (Quit) เพื่อออกจาก Turbo Pascal เข้าสู่ DOS

ขั้นตอนการเขียนโปรแกรมภาษา Pascal ถ้าเขียนเป็น **flowchart** จะเขียนได้เป็นลักษณะดังต่อไปนี้



2.2 คำสั่งอินพุทและเอาท์พุท

เมื่อต้องการจะสื่อกับคอมพิวเตอร์ จำเป็นจะต้องเข้าใจคำสั่งที่ใช้ในการสื่อ ในบทนี้จะกล่าวถึงคำสั่งและโปรแกรมที่มีคำสั่งที่ใช้ในการรับส่งข้อมูลกับคอมพิวเตอร์ ซึ่งนับว่าเป็นพื้นฐานของการเขียนโปรแกรม

ภาษา Pascal คำสั่งอินพุทเอาท์พุทมีอยู่ 4 คำสั่ง ดังตาราง

	คำสั่ง	การดำเนินการ
อินพุท	read	รับข้อมูลจากคีย์บอร์ด
	readln	รับข้อมูลจากคีย์บอร์ด เมื่อรับข้อมูลแล้ว เคอร์เซอร์จะเขยิบลงมาหนึ่งบรรทัด
เอาท์พุท	write	ส่งข้อมูลหรือคำตอบมาทางจอ
	writeln	เหมือน write แต่เมื่อส่งข้อมูลเสร็จ จะขึ้นบรรทัดใหม่

2.3 คำสั่ง write / writeln

คำสั่ง write และ writeln เป็นคำสั่งที่ส่งเอาท์พุทออกมา ซึ่งมีแบบการเขียนดังต่อไปนี้คือ

```
write(พารามิเตอร์ [:ความกว้าง],พารามิเตอร์ [:ความกว้าง] ...)
writeln(พารามิเตอร์ [:ความกว้าง],พารามิเตอร์ [:ความกว้าง] ...)
```

พารามิเตอร์ในวงเล็บ อาจเป็นตัวแปร ตัวคงที่ ตัวอักขระ หรือนิพจน์ต่างๆ เช่น a+b หรือ a div b เป็นต้น การกำหนดความกว้างนั้นอาจจะมีหรือไม่ก็ได้ ในกรณีที่มีการกำหนดความกว้างจะเป็นการจัดช่องแสดงเอาท์พุทให้อยู่ชิดทางขวาของความกว้างที่ได้กำหนด ถ้าหากมีช่องเหลือก็จะเป็นที่ว่าง ถ้าเอาท์พุทเป็นเลขจำนวนเต็มก็เพียงแต่กำหนดความกว้างเพียงอันเดียว แต่ถ้าเป็นเลขจำนวนจริง อาจกำหนดว่าต้องการทศนิยมกี่ตำแหน่งด้วยก็ได้ ดังตัวอย่าง

เลขจำนวนจริง : ความกว้าง : ตำแหน่งทศนิยมที่ต้องการ

คำสั่ง writeln ก็สามารถเขียนได้เช่นเดียวกับ write เมื่อส่งเอาท์พุทออกมาแล้วจะขึ้นบรรทัดใหม่ และคำสั่ง writeln สามารถเขียนโดยไม่ต้องมีพารามิเตอร์อยู่ข้างในวงเล็บ ซึ่งหมายความถึงการขึ้นบรรทัดใหม่ธรรมดา

```
writeln(พารามิเตอร์:ความกว้าง) ------------------>   ______[    ]
                                                     <- ความกว้าง ->

writeln(เลขจำนวนจริง:ความกว้าง:ตำแหน่งทศนิยม) --->   ____[   ].[   ]
                                                            ทศนิยม
                                                     <- ความกว้าง ->
```

```
ex  write('a'); ------------>         a
    writeln('a':5); -------->         ____a
    writeln(100);   -------->         100
    writeln(100:6); -------->         ___100
    writeln(100.5); -------->         _1.0050000000E+02
    writeln(100.5:10); ----->         1.0050E+02
    writeln(100.5:10:2); --->         ____100.50
    write('he'); ----------->         hello
    write('llo'); ------|--->         ขึ้นบรรทัดใหม่
    writeln; -----------
```

ตัวอย่างที่ 2.1 ตัวอย่างแสดงตำแหน่งเอาท์พุทของการคำนวณเลขจำนวนเต็ม

```
program test_write;
var a:integer;
    b:real;
begin
   a:=12345;
   b:=100.5;
   writeln('....+....+....+....');
   writeln(a);                        {integer}
   writeln(a:10);
   writeln(b);                        {real}
   writeln(b:10:3)
end.
```

```
....+....+....+....
12345
     12345
 1.0050000000E+02
   100.500
```

ตัวอย่างที่ 2.2 เขียนคำว่า "PASCAL" ตัวใหญ่

```
program WritePascal;
begin
   writeln('PPPPP      A      SSSSS     CCCCC     A     L      ');
   writeln('P    P    A A    S         C         A A    L      ');
   writeln('P    P   A   A   S         C        A   A   L      ');
   writeln('PPPPP   AAAAAAA   SSSSS    C       AAAAAAA  L      ');
   writeln('P       A     A        S   C       A     A  L      ');
   writeln('P       A     A        S   C       A     A  L      ');
   writeln('P       A     A   SSSSS     CCCCC  A     A  LLLLLL')
end.
```

ตัวอย่างที่ 2.3 เอาท์พุทของคำสั่ง writeln แบบกำหนดความกว้าง

```
program test_write;
var a:integer;
   b:real;
   c:char;
begin
   a:=12345;
   b:=1.23456789E-1;
   c:='*';
   writeln('a=',a);
   writeln('a=',a:10);

   writeln('b=',b);
   writeln('b=',b:10);
   writeln('b=',b:12);
   writeln('b=',b:20);
   writeln('b=',b:10:5);
   writeln('b=',b:12:5);
   writeln('b=',b:12:8);

   writeln('c=',c);
   writeln('c=',c:10);
end.
```

```
a=12345
a=     12345
b= 1.2345678900E-01
b= 1.235E-01
b= 1.23457E-01
b=     1.2345678900E-01
b=   0.12346
b=      0.12346
b=  0.12345679
c=*
c=         *
```

เมื่อสังเกตให้ดี จะเห็นว่าในกรณีที่มีการกำหนดขอบเขตการแสดงเอาท์พุทของเลขจำนวนจริง จะมีการปัดขึ้นถ้าหากเกิน 5 ยกตัวอย่างเช่น

write(1.2385:10:2)

ผลที่ได้จะเป็น ______1.24

2.4 คำสั่ง read / readln

read และ readln เป็นคำสั่งที่ใช้สำหรับการรับข้อมูลจากคีย์บอร์ด โดยมีแบบของการเขียนดังต่อไปนี้

read(พารามิเตอร์,พารามิเตอร์,...)

readln(พารามิเตอร์,พารามิเตอร์,...)

สำหรับพารามิเตอร์ในวงเล็บ จะใส่ตัวแปรได้เพียงอย่างเดียว ไม่สามารถใส่นิพจน์หรือกำหนดความกว้างเหมือนคำสั่ง write หรือ writeln

การใส่อินพุทนั้น ในกรณีที่มีอินพุทมากกว่าหนึ่งตัว ต้องใส่เว้นวรรค (space) ระหว่างข้อมูลด้วย เมื่อใส่อินพุทเรียบร้อยแล้ว ก็กด [enter] เพื่อให้เครื่องรับข้อมูล คำสั่ง read กับคำสั่ง readln จะแตกกันเล็กน้อย คือ คำสั่ง readlnจะมีปฏิกิริยาตอบมา คือ เคอร์เซอร์จะกระเถิบลงมาหนึ่งบรรทัด ส่วนคำสั่ง read เครื่องจะรับข้อมูลโดยไม่มีปฏิกิริยาใดๆเลย คำสั่ง readln เป็นคำสั่งสามารถเขียนโดยไม่ใส่พารามิเตอร์ก็ได้ การดำเนินการก็คือ รอรับ [enter] จากผู้ใช้

read(ตัวแปร,ตัวแปร,...)

10 50 [enter]

ช่องว่าง จบการส่งข้อมูล

ex

```
var  a,b:integer;      (เลขจำนวนเต็ม)
     x,y:real;         (เลขจำนวนจริง)
     c,d:char;         (อักขระ)
     ....
                              10 20 [enter]
     readln(a,b);
                              100.5 200 [enter]
     readln(x,y);
                              *! [enter]
     readln(c,d);
```

ตัวอย่างที่ 2.4 ใส่ค่าให้กับเลขจำนวนเต็ม และทำการบวกลบคูณหาร และหาเศษของเลขทั้งสอง

```
program calculate1;
var a,b,c,d,e,f,g:integer;
begin
  write(' a= ');read(a);
  write(' b= ');readln(b);
  c:=a+b; d:=a-b;
  e:=a*b; f:=a div b;
  g:=a mod b;
  writeln(c:10,d:10,e:10,f:10,g:10)
end.
```

```
a= 50 [enter]
b= 6  [enter]
        56        44       300         8        2
```

ตัวอย่างที่ 2.5 ใส่ค่ารัศมี และทำการคำนวณหาพื้นที่วงกลมและปริมาตรทรงกลม

```
program calculate2;
const pi=3.14; ....................................... 1
var radius,area,cube:real;
begin
   write('radius=');
   readln(radius);
   area:=pi*radius*radius;................................ 2
   cube:=4/3*pi*radius*radius*radius; .............. 3
   writeln('area =',area:10:3);
   writeln('cube =',cube:10:3)
end.
```

```
radius=3 [enter]
area =    28.260
cube =   113.040
```

1 กำหนดค่าคงที่

2 คำนวณหาพื้นที่

3 คำนวณหาปริมาตร

Coffee Break

ในภาษา Pascal โอเปอร์เรเตอร์ที่ดำเนินการหารมีอยู่ 2 ชนิด คือ / และ div การใช้โอเปอร์เรเตอร์ทั้งสองต้องใช้ด้วยความระมัดระวัง เพราะอาจเกิด ERROR ขึ้นได้ จากตัวอย่าง การคำนวณที่มี * จะเกิด ERROR ขึ้น

```
    program test_operators;
    var x,y,z:integer;
        r,s,t:real;
    begin
       x:=1;y:=2;s:=3;t:=4;
       z:=x div y;
*      z:=s div y;
       r:=x div y;
*      z:=x / y;
       r:=x / t;
       r:=s / t;
    end.
```

ความแตกต่างของการใช้ / กับ div คือ

div ใช้กับการคำนวณระหว่างเลขจำนวนเต็ม

/ ใช้กับการคำนวณระหว่างเลขจำนวนจริง หรือการคำนวณแบบผสมระหว่างเลขจำนวนเต็มกับเลขจำนวนจริง โดยผลที่ได้จะเป็นเลขจำนวนจริง

บทที่ 3

โครงสร้างภาษา Pascal

3.1 โครงสร้างภาษา Pascal

ภาษา Pascal มีโครงสร้างทันสมัยมีการจัดโครงสร้างอย่างเป็นระบบ โดยมีหลักเกณฑ์และคำสั่งดังต่อไปนี้

1 ลำดับ (sequence)

ข้อความ S_1 S_1;

ข้อความ S_2 S_2;

หลักเกณฑ์ขั้นแรกที่ควรทำความเข้าใจคือ ลำดับ (sequence) การดำเนินการจะเริ่มจากบรรทัดแรกไปจนถึงบรรทัดสุดท้ายตามลำดับ (S_1 S_2 ...)

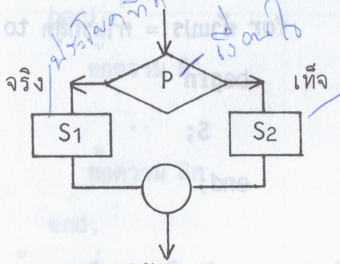
2 การให้ค่าตัวแปร

ex a:=1; (ให้ a เท่ากับ 1 การให้ค่า ตัวแปรต้องอยู่ทางซ้าย)

b:=a; (ให้ b เท่ากับ a ซึ่งเป็นการอ้างค่า a การอ้างค่าตัวแปรต้องอยู่ทางขวา)

c:=c+1; (เพิ่ม c ขึ้นจากเดิมอีก 1 เป็นการให้ค่า c โดยอ้างค่าเดิมของ c)

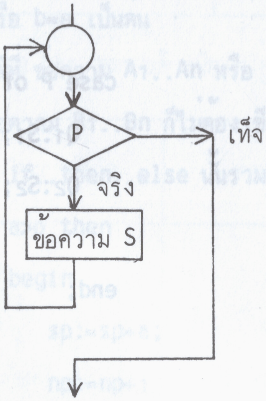
3 เงื่อนไข (if..then..else)




```
if P then
    S1
else
    S2;
```

ดำเนินการข้อความ S_1 หรือ S_2 ขึ้นอยู่กับเงื่อนไข (P) เป็นจริงหรือเท็จ

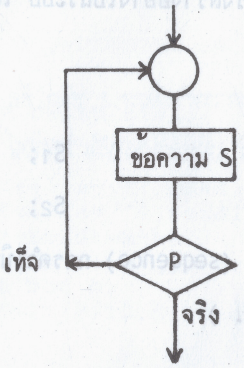
4 การวนซ้ำแบบกำหนดเงื่อนไขตอนต้น (while..do)




```
while P do
begin
    S;
end;
```

เงื่อนไข (P) อยู่ตอนต้น

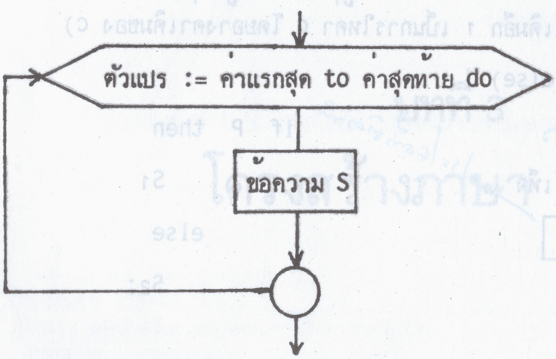
5 การวนซ้ำแบบกำหนดเงื่อนไขตอนท้าย (repeat..until)




```
repeat
    S;
until P;
```

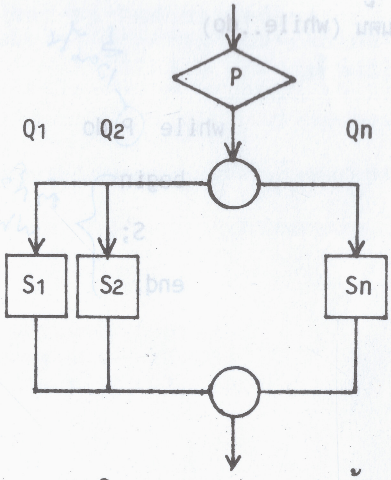
เงื่อนไข (P) อยู่ตอนท้าย

6 การวนซ้ำแบบกำหนดจำนวนครั้ง (for..to..do)




```
for ตัวแปร:= ค่าแรกสุด to ค่าสุดท้าย
    begin
      S;
    end;
```

7 การดำเนินแยกตามกรณี (case..of)




```
case P of
    Q1:S1;
    Q2:S2;

end;
```

ดำเนินการ S_1, S_2 ... Sn ข้อความใดข้อความหนึ่ง โดยขึ้นกับ P ว่ามีค่าเท่าใด

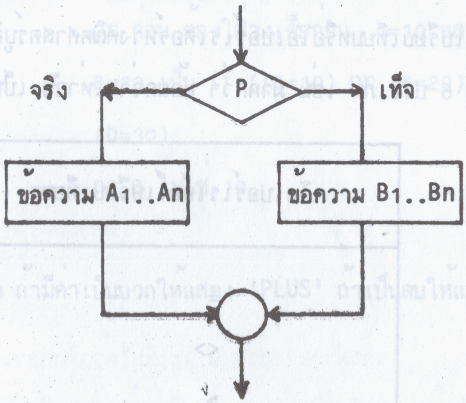
๓.๒ คำสั่งดำเนินการตามเงื่อนไข

การเขียนโปรแกรมทุกโปรแกรมต้องมีคำสั่งเงื่อนไขเข้ามาเกี่ยวข้อง โดยทำการตั้งเงื่อนไขขึ้นมา และเขียนโปรแกรมดำเนินการตามเงื่อนไขทั้งในกรณีที่เงื่อนไขนั้นเป็นจริงและเท็จ โดยใช้คำสั่ง if..then..else

< คำสั่ง if..then..else >

คำสั่ง if..then..else สามารถเขียนเป็นลักษณะดังต่อไปนี้

```
if เงื่อนไข then
   begin
      ข้อความ A1
        ..
      ข้อความ An
   end   ห้ามใส่ ;
else
   begin
      ข้อความ B1
        ..
      ข้อความ Bn
   end;
```



ถ้าหากเงื่อนไขเป็นจริง จะดำเนินการข้อความ $A_1..A_n$ ถ้าเป็นเท็จก็ดำเนินการข้อความ $B_1..B_n$ สำหรับการเขียนเงื่อนไขนั้น เขียนเหมือนกับการให้ค่าแก่ตัวแปร แต่ไม่มี : ก่อน = เช่น a=0 หรือ b=8 เป็นต้น

ในกรณีที่ ข้อความ $A_1..A_n$ หรือ $B_1..B_n$ มีเพียงหนึ่งข้อความก็ไม่ต้องมี begin..end และถ้าหากไม่มีข้อความ $B_1..B_n$ ก็ไม่ต้องเขียนข้อความหลัง else

คำสั่ง if..then..else นั้นรวมกันเป็นหนึ่งข้อความ จึงใช้ ; เพียงที่สุดท้ายที่เดียว

```
ex    if a>0 then
        begin
           sp:=sp+a;
           np:=np+1          (* ไม่จำเป็นต้องมี ; ก่อน end ก็ได้ *)
        end                  (* ห้ามใส่ ; *)
```

```
else
  begin
    sm:=sm+a;
    nm:=nm+1
  end;
```

(วางไว้ ; หรือไม่ก็ได้)

จากตัวอย่างข้างบน ถ้า a มากกว่า 0 ให้ sp=sp+a และ np=np+1 ถ้า a น้อยกว่าหรือเท่ากับ 0 ให้ sm=sm+a และ nm=nm+1

< โอเปอร์เรเตอร์ที่ใช้ในการเปรียบเทียบและโอเปอร์เรเตอร์ทางคณิตศาตร์บูลีน >

คำสั่ง if หรือคำสั่ง while (ซึ่งจะอธิบายภายหลัง) จะต้องมีโอเปอร์เรเตอร์ที่ใช้ในการเปรียบเทียบหรือโอเปอร์เรเตอร์ทางคณิตศาสตร์บูลีน โอเปอร์เรเตอร์ที่ใช้ในการเปรียบเทียบมีอยู่ 6 ประเภท เช่น มากกวา นอยกวา เทากับ เป็นต้น

โอเปอร์เรเตอร์เปรียบเทียบ	ความหมาย
=	เท่ากับ
<>	ไม่เท่ากับ
>	มากกว่า
>=	มากกว่าหรือเท่ากับ
<	น้อยกว่า
<=	น้อยกว่าหรือเท่ากับ

โอเปอร์เรเตอร์ทางคณิตศาตร์บูลีน มีอยู่ 3 ชนิด คือ not and และ or ดังตาราง

โอเปอร์เรเตอร์	ความหมาย	ลำดับการดำเนินการ
not	ไม่	ก่อน
and	และ	↓
or	หรือ	หลัง

นอกจากนี้ใน Turbo Pascal ยังมีโอเปอร์เรเตอร์ xor อีกด้วย ซึ่งผลเป็น true เมื่อทั้งสองข้างของโอเปอร์เรเตอร์ xor ไม่เหมือนกัน (เช่น true กับ false หรือกลับกัน)

ในกรณีที่ใช้โอเปอร์เรเตอร์หลายตัวในข้อความเดียวกัน ต้องใส่วงเล็บครอบส่วนที่ต้องการให้ดำเนินการก่อน มิเช่นนั้น การดำเนินการจะทำตามลำดับ not and และ or หรือ xor

ex

if (a>b) and (b>c) ---> ถ้า a>b และ b>c

if (a=10) or (a=20) ---> ถ้า a=10 หรือ a=20

if (a=10) or (a=20) and (b=30) ---> การดำเนินการจะดำเนินการโอเปอร์เรเตอร์ and ก่อน or ในกรณีที่ต้องการดำเนินคำสั่ง or ก่อน ต้องใส่วงเล็บก่อน a=10 และหลัง a=20 เป็น if ((a=10) or (a=20))and (b=30)

if not (a=10) ---> ถ้า a ไม่ใช่ 10

ตัวอย่างที่ 3.1 ใส่เลขจำนวนเต็ม ถ้ามีค่าเป็นบวกให้แสดง 'PLUS' ถ้าเป็นลบให้แสดง 'MINUS'

```
program if_else;
var a:integer;
begin
   write('data ? ');
   readln(a); ...................... 1
   if a>=0 then .................... 2
      writeln('plus')
   else
      writeln('minus')
end.

data ? 2 [enter]
plus
```

1 ให้ค่า a

2 การเขียนเงื่อนไขอาจเขียนเป็นลักษณะ

if a>= 0 then writeln('PLUS')

else writeln('MINUS') ก็ได้

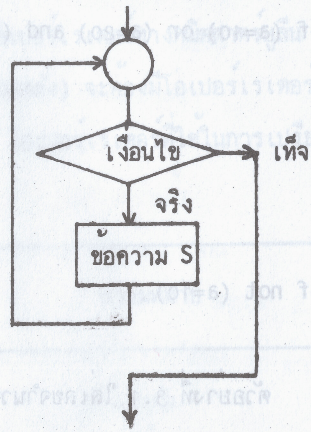
3.8 การวนซ้ำแบบกำหนดเงื่อนไขตอนต้น

การวนซ้ำ (หรือเรียกว่า ลูป [loop]) เป็นคำสั่งที่สำคัญที่สุดของโครงสร้างภาษาคอมพิวเตอร์ สำหรับภาษา Pascal มีคำสั่งวนซ้ำดังต่อไปนี้คือ while..do ,for..to..do และ repeat..until ก่อนอื่นจะกล่าวถึงการใช้คำสั่ง while..do ก่อน

< คำสั่ง while..do >

while..do เป็นคำสั่งวนซ้ำโดยมีการกำหนดเงื่อนไขของการวนอยู่ตอนต้นซึ่งสามารถเขียนเป็นลักษณะทั่วไปได้ดังนี้

```
while  เงื่อนไข  do
  begin
       ..
       ข้อความ S
       ..
  end;
```



จากตัวอย่างข้างบน จะเห็นได้ว่ามีการวนข้อความ S ซ้ำ จนกระทั่งเงื่อนไขเป็นเท็จ สำหรับข้อความ S นั้น ถ้าหากเป็นเพียงข้อความเดียวก็ไม่จำเป็นต้องมี begin กับ end

วิธีการเขียนเงื่อนไขจะมีลักษณะเหมือนคำสั่ง if..then

ex
```
readln(data);           <--+
while data <> -999 do      |
  begin                    |-- คำสั่งอ่าน 2 ครั้ง
    sum:=sum+data;         |
    readln(data)        <--+
  end;
```

คำสั่ง while .. do เป็นคำสั่งที่สามารถใช้ในการอ่านข้อมูลที่ไม่ได้กำหนดจำนวน จากตัวอย่างจะเห็นได้ว่า จะมีการวนซ้ำไปจนกว่าข้อมูลเป็น -999 ซึ่ง sum ก็จะเป็นผลรวมของข้อมูลที่ใส่เข้าไป สำหรับสาเหตุของการเขียนคำสั่ง read 2 คำสั่งคือ ถ้าหากไม่มีคำสั่ง read ตัวแรกแล้ว ในกรณีที่มีการดำเนินการไปก่อนแล้ว data สุดท้าย (-999) จากการดำเนินจะอยู่ในหน่วยความจำซึ่งทำให้การดำเนินการครั้งต่อไปไม่เข้าไปในลูป while..do

ตัวอย่างที่ 3.2 โปรแกรมเก็บข้อมูลชุดหนึ่ง และคำนวณหาผลรวม ค่าเฉลี่ย ค่ามากสุด และค่าน้อยสุดของข้อมูลชุดนี้ โดยกำหนดให้ข้อมูลที่มีค่า -999 เป็นเงื่อนไขในการสิ้นสุดการส่งข้อมูล (-999 ไม่ถือเป็นข้อมูล)

< วิธีคิด > จุดสำคัญของโปรแกรมนี้อยู่ที่การหาค่ามากสุดและน้อยสุด ตัวแปรที่ใช้แทนค่ามากสุดและน้อยสุดคือ max และ min ตามลำดับ เมื่อใส่ข้อมูลเข้าไป ถ้าหากข้อมูลใหม่นี้มีค่ามากกว่า max ก็ให้ค่า max มีค่าเท่ากับข้อมูลใหม่ ในทำนองเดียวกัน ถ้าข้อมูลใหม่มีค่าน้อยกว่า min ก็ให้ min มีค่าเท่ากับข้อมูลใหม่ โดยกำหนดในตอนแรกสุดให้ทั้ง max และ min มีค่าเป็นข้อมูลแรก

```
program while_do;
var data,sum,n,min,max:integer;
begin
   sum:=0;  n:=0;  ............................ 1
   readln(data);  ............................ 2
   min:=data;  max:=data;
   while data<>-999 do  ...................... 3
      begin
         sum:=sum+data;
         n:=n+1;
         if data<min then min:=data;
         if data>max then max:=data;
         readln(data)
      end;
   writeln('sum = ',sum);
   writeln('mean = ',sum div n);  ............ 4
   writeln('maximum is ',max);
   writeln('minimum is ',min)
end.
```

```
    22    [enter]
    55    [enter]
    21    [enter]
    65    [enter]
    78    [enter]
    -999  [enter]
    sum = 241
    mean = 48
    maximum is 78
    minimum is 21
```

1 ให้ค่าเริ่มต้นแก่ตัวแปรที่ใช้เป็นผลรวม (sum) และจำนวนข้อมูล (n)

2 อ่านข้อมูลตัวแรก

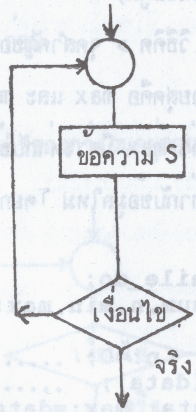
3 รับข้อมูลจนกว่าข้อมูลจะมีค่าเป็น -999 และให้ค่า max และ min ใหม่

4 การแสดงผล (ใช้ div แทน / เพราะเป็นเลขจำนวนเต็ม)

3.4 การวนซ้ำแบบกำหนดเงื่อนไขที่ตอนท้าย

คำสั่ง repeat..until เป็นคำสั่งการวนซ้ำ โดยการวนมีวิธีกำหนดเงื่อนไขตรงข้ามกับคำสั่ง while..do สามารถเขียนเป็นลักษณะทั่วไปได้ดังนี้

```
repeat
    ข้อความ S
    ..
    ..
until เงื่อนไข
```



คำสั่งดังกล่าวจะดำเนินการ จนกระทั่งเงื่อนไขเป็นจริง คำสั่งนี้ไม่ต้องมี begin..end เหมือนคำสั่ง while..do ไม่ว่าข้อความ S จะมากกว่าหนึ่งหรือไม่ก็ตาม

การดำเนินการของคำสั่ง repeat..until ต่างกับ while..do เล็กน้อย กล่าวคือ เมื่อเงื่อนไขเป็นจริงก็ยังทำการวนซ้ำอีกครั้งหนึ่งก่อนที่จะออกนอกลูป ดังนั้น ในบางกรณีอาจเกิดการผิดพลาด เช่น ถ้าสมมุติอ่านค่า -999 เป็นค่าแรก ก็จะมีการดำเนินการข้อความในลูป ดังนั้นจึงควรคำนึงถึงความแตกต่างนี้และพิจารณาคู่กับคำสั่ง while..do เพื่อให้เหมาะกับวัตถุประสงค์

```
ex  readln(data);
    repeat
      sum:=sum+data;
      readln(data);        คำสั่งอ่าน 2 ครั้ง
    until data=-999
```

การวนซ้ำจะดำเนินการไปจนกระทั่ง data =-999 จึงจะออกนอกลูป และในระหว่างนั้นก็จะมีการบวกข้อมูลทุกข้อมูลเข้าที่ตัวแปร sum (ยกเว้น -999) สำหรับคำสั่ง readln 2 คำสั่ง มีเหตุผลเช่นเดียวกับคำสั่ง while..do

ดังนั้น ตัวอย่างที่ 3.2 อาจแก้ไขคำสั่ง while..do มาเป็นคำสั่ง repeat..until และทำการดำเนินการเหมือนเดิมได้ทุกประการ โดยเปลี่ยนคำสั่งบางคำสั่งดังต่อไปนี้

```
while data <> -999 do begin  ------>  repeat
end                          ------>  until data=-999
```

ตัวอย่าง 3.3 โปรแกรมแสดงการเปรียบเทียบว่า A กับ B ด้วยโอเปอร์เรเตอร์ โดยให้แก่ตัวแปร A และ B (A และ B เป็นเลขจำนวนเต็ม)

```
program compare;
var a,b:real;
   c:char;
begin
   repeat
      write('A=');
      readln(a);                    ......................................... 1
      write('B=');
      readln(b);
      writeln('the result of A=B is  ',a=b);
      writeln('the result of A<>B is  ',a<>b);
      writeln('the result of A>B is   ',a>b);
      writeln('the result of A>=B is  ',a>=b); .......... 2
      writeln('the result of A<B is   ',a<b);
      writeln('the result of A<=B is  ',a<=b);
      write('Compare again ?(y/n)  '); .................... 3
      readln(c);
   until  (c='N') or (c='n');
end.
```

```
A=-23  [enter]
B=33   [enter]
the result of A=B is   FALSE
the result of A<>B is TRUE
the result of A>B is  FALSE
the result of A>=B is FALSE
the result of A<B is  TRUE
the result of A<=B is TRUE
Compare again ?(y/n)  n  [enter]
```

1 อ่านค่า A และ B

2 ถ้า A มากกว่า B   A>B จะเป็น TRUE

ถ้า A น้อยกว่า B   A>B จะเป็น FALSE

โอเปอร์เรเตอร์อื่นก็ใช้หลักการเดียวกัน

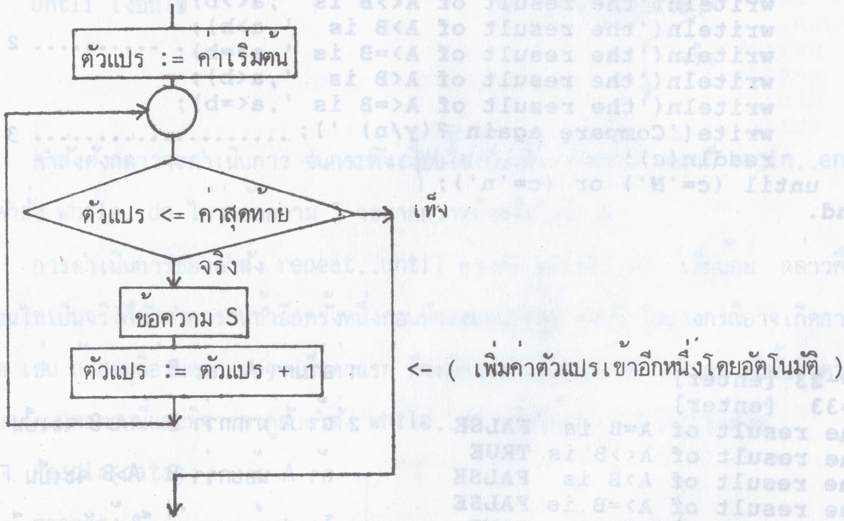
3 ถามผู้ใช้ว่าต้องการดำเนินการโปรแกรมนี้อีกหรือไม่ ถ้าไม่ต้องการก็ใส่ n,N คำสั่ง repeat..(c='N') or (c='n') เหมาะสำหรับ โปรแกรมที่ต้องการวนซ้ำจนกว่าผู้ใช้พอใจ (ถ้าต้องการดำเนินการอีกใส่ตัวอักษรอื่น)

3.5 การวนซ้ำแบบกำหนดจำนวนครั้ง

< คำสั่ง for..to..do >

คำสั่ง for เป็นคำสั่งที่ใช้ในการดำเนินการวนซ้ำ โดยสามารถกำหนดจำนวนครั้งของการวนซ้ำได้ ลักษณะทั่วไปสามารถเขียนได้ดังนี้

```
for ตัวแปร := ค่าเริ่มต้น  to  ค่าสุดท้าย do
begin
    ข้อความ S
.........................................
end;
```



จะเห็นได้ว่า ระหว่างที่ตัวแปรเปลี่ยนจากค่าแรกไปจนถึงค่าสุดท้าย มีการดำเนินการข้อความ S สำหรับแบบตัวแปรที่ใช้นั้น นอกจากเลขจำนวนเต็มแล้ว สามารถใช้ตัวแปรแบบสเกลาร์ (ดูบทที่ 5) ได้ทุกชนิด ในกรณีที่ต้องการเพิ่มค่าหรือลำดับตัวแปรทีละ 1 ก็ใช้ to และใช้ downto ในกรณีที่ต้องการลดค่าหรือลำดับตัวแปรทีละ 1

ตัวแปรที่ใช้เป็นคำสั่งลูปห้ามนำไปให้ค่าในลูป (ในข้อความ S) สำหรับค่าเริ่มต้น และค่าสุดท้าย นอกจากตัวเลขแล้วยังสามารถใช้ตัวแปรหรือนิพจน์ได้ด้วย

ในกรณีที่ข้อความ S มีเพียงหนึ่งข้อความก็ไม่จำเป็นจะต้องมี begin..end

คำสั่ง for ในภาษา Pascal นั้น มีจุดที่ไม่สะดวกที่ไม่สามารถเพิ่มค่าตัวแปรได้ทีละมากกว่าหรือน้อยกว่าหนึ่ง ในกรณีเช่นนี้ต้องดัดแปลงคือใช้ตัวแปรเพิ่มอีกหนึ่งตัว หรือใช้คำสั่ง while do,repeat..until แทน

```
ex   for i:=1 to 5 do       ......   ค่า i จะเพิ่มจาก -5 เป็น 5 และในเวลา
       begin                         เดียวกันก็จะดำเนินการข้อความภายในที่อยู่
                                     ภายใน begin กับ end
       end;

     for j:=k+1 downto 0    ......   ค่า j จะลดจาก k+1 ไปจนถึง 0 ทีละ 1
       begin                         ในระหว่างนั้นจะดำเนินการข้อความภายใน
                                     begin กับ end (ถ้า k+1 มีค่าน้อยกว่า
       end;                          0 จะไม่มีการเข้าไปในลูป)
```

ตัวอย่างที่ 3.4 เขียนอักขระตามรหัส ASCII ตั้งแต่ 32 ไปจนถึง 127

```
program for_to_do;
var i:integer;
begin
   for i:=32 to 127 do  ......................... 1
      begin
         if (i mod 8)=0 then writeln;  ......... 2
         write(i:4,chr(i):2)           ......... 3
      end
   end.
```

```
 32     33 !   34 "   35 #   36 $   37 %   38 &   39 '
 40 (   41 )   42 *   43 +   44 ,   45 -   46 .   47 /
 48 0   49 1   50 2   51 3   52 4   53 5   54 6   55 7
 56 8   57 9   58 :   59 ;   60 <   61 =   62 >   63 ?
 64 @   65 A   66 B   67 C   68 D   69 E   70 F   71 G
 72 H   73 I   74 J   75 K   76 L   77 M   78 N   79 O
 80 P   81 Q   82 R   83 S   84 T   85 U   86 V   87 W
 88 X   89 Y   90 Z   91 [   92 \   93 ]   94 ^   95 _
 96 `   97 a   98 b   99 c  100 d  101 e  102 f  103 g
104 h  105 i  106 j  107 k  108 l  109 m  110 n  111 o
112 p  113 q  114 r  115 s  116 t  117 u  118 v  119 w
120 x  121 y  122 z  123 {  124 |  125 }  126 ~  127
```

1 เพิ่มค่าตัวแปรตัวแปร i จาก 32 ไปจนถึง 127

2 ขึ้นบรรทัดใหม่ทุกๆ 8 ตัวอักขระ

3 chr(i) หมายถึง อักขระตัวที่ i ของ ASCII CODE (รายละเอียดดูบทที่ 9)

< การวนซ้ำแบบซ้อนกัน >

คำสั่งการวนซ้ำสามารถเขียนซ้อนกันได้ ซึ่งมักจะใช้กับคำสั่ง for..to..do โดยการดำเนินการจะเริ่มจากลูปในก่อน

```
ex    for i:=1 to 10 do
          begin
              for j:=1 to 10 do
                  begin      }
                             }  ดำเนินการลูปในก่อน
                  end;       }

          end;
```

ตัวอย่างที่ 3.5 การเขียนสูตรคูณ (1 ถึง 7)

```
program double_loop;
const n=7; ................................ 1
var j,k:integer;
begin
  for j:=1 to n do
     begin
        for k:=1 to n do
           write(k*j:5); ................... 2
        writeln  ........................... 3
     end
end.
```

1	2	3	4	5	6	7
2	4	6	8	10	12	14
3	6	9	12	15	18	21
4	8	12	16	20	24	28
5	10	15	20	25	30	35
6	12	18	24	30	36	42
7	14	21	28	35	42	49

1 กำหนดค่าคงที่ให้เป็น 7

2 เขียนค่า j*k โดยเริ่มจากค่า j=1 และเริ่มดำเนินการวนลูป for k:=1 to n

3 ขึ้นบรรทัดใหม่ทุกครั้งที่ค่า j เปลี่ยน

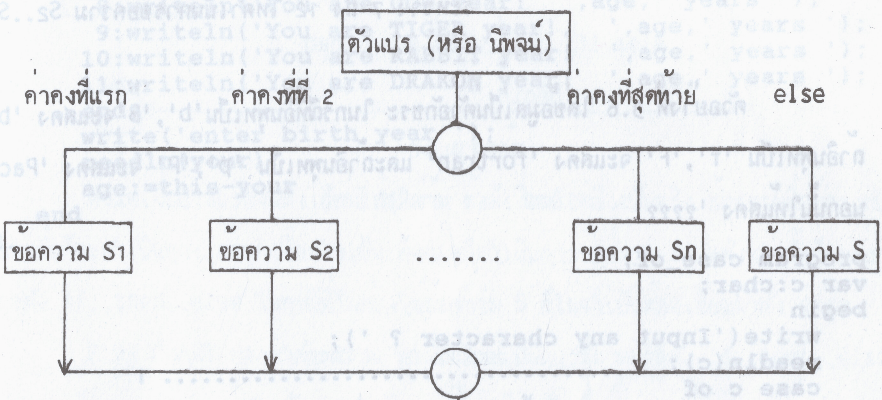
3.6 คำสั่งดำเนินการแยกตามกรณี

คำสั่ง if..then เป็นคำสั่งเลือกการดำเนินการทางใดทางหนึ่ง ซึ่งขึ้นอยู่กับเงื่อนไขหลังคำสั่ง if เป็นจริงหรือเท็จ นอกจากคำสั่ง if..then..else แล้ว อาจใช้คำสั่ง case..of ได้เช่นกัน

< คำสั่ง case..of >

คำสั่ง case..of สามารถเขียนเป็นลักษณะทั่วไปได้ดังนี้

```
case ตัวแปร (หรือ นิพจน์) of
    ค่าคงที่แรก    : ข้อความ S1;
    ค่าคงที่สุดท้าย : ข้อความ Sn;
else ข้อความ S    (* ใน Standard Pascal ใช้ else หลัง case ไม่ได้ *)
end;              (* คำสั่ง case ต้องคู่กับ end เสมอ *)
```



เมื่อนิพจนหรือตัวแปรมีค่าตรงกับค่าคงที่ใด ก็จะดำเนินการขอความที่ตรงกับค่าคงที่นั้น ในกรณีที่ไม่ตรงกับค่าคงที่ใดๆเลย ก็จะดำเนินการข้อความหลัง else (ขอความ S) ซึ่งข้อความนี้อาจไม่เขียน ถ้าไม่จำเป็น

นิพจน์หรือตัวแปรที่ใช้กับคำสั่ง case..of สามารถใช้กับข้อมูลแบบสเกลาร์ (บทที่5) แต่ใช้เลขจำนวนจริงไม่ได้ และถ้าใช้คอมมาคั่นระหว่างค่าคงที่ก็มีความหมายเหมือนเงื่อนไข or

ระหว่างค่าคงหรือนิพจน์กับข้อความต้องมีโคลอน (:) คั่น และถ้ามีมากกว่า 1 ข้อความ ต้องมี begin กับ end ปิดหัวท้ายบล็อคนั้นด้วย นอกจากนี้ คำสั่ง case..of ยังสามารถใช้ซ้อนกันเป็นคำสั่ง case..of ในคำสั่ง case..of ได้อีกด้วย

```
ex    case c of
         'h','H'  :writeln('hello');
         '!'      :writeln('good bye');
      end;             .... ถ้าตัวแปรแบบอักขระ c เป็น 'h' หรือ 'H' ให้แสดง
                            'hello' ถ้าเป็น '!' ให้แสดง 'good bye'
      case month of
         1..6 : ข้อความ S1              (* ข้อความเดียว *)
         7..12: bigin
                  ข้อความ S2..Sn       (* มากกว่า 1 ข้อความ *)
                end;
      end;             .... ตัวแปร month เป็นข้อมูลแบบเลขจำนวนเต็ม ถ้ามีค่าอยู่
                            ระหว่าง 1 ถึง 6 ให้ดำเนินการข้อความ S1 ถ้ามีค่าอยู่
                            ระหว่าง 7 ถึง 12 ให้ดำเนินการข้อความ S2..Sn
```

ตัวอย่างที่ 3.6 ใส่ข้อมูลเป็นตัวอักขระ ในกรณีที่อินพุทเป็น'b','B'จะแสดง 'basic' ถ้าอินพุทเป็น 'f','F' จะแสดง 'fortran' และถ้าอินพุทเป็น 'p','P' จะแสดง 'Pacsal' นอกนั้นให้แสดง '????'

```
program case_of;
var c:char;
begin
  write('Input any character ? ');
  readln(c); ..................................... 1
  case c of
     'b','B' :writeln('basic');
     'f','F' :writeln('fortran');
     'p','P' :writeln('Pascal'); .............. 2
    else writeln('???')...................... 3
  end
end.
```

```
  p [enter]                          1 อ่านอักขระหนึ่งตัว
  Pascal                             2 ค่าของตัวแปร c เป็น 'p' จึงดำเนิน
                                       การข้อความ writeln('Pascal')
                                     3 ห้ามใส่ ; หลัง else
```

ตัวอย่างที่ 3.7 โปรแกรม 12 ราศี โดยใส่ข้อมูลปีปัจจุบันและปีเกิดเป็น พ.ศ.และแสดงผลเป็นราศีที่เกิดและอายุปัจจุบัน

```
program Zodiac_signs;
var this,your,age:integer;
begin
   write('enter this year ');
   readln(this); ................................ 1
   write('enter birth year ');
   read(your); ................................... 2
   age:=this-your;
   while your<>0 do
      begin
         case your mod 12 of ..................... 3
         0:writeln('You are SNAKE year!   ',age,' years ');
         1:writeln('You are HORSE year!   ',age,' years ');
         2:writeln('You are SHEEP year!   ',age,' years ');
         3:writeln('You are MONKEY year!  ',age,' years ');
         4:writeln('You are HEN year!   ',age,' years ');
         5:writeln('You are DOG year!   ',age,' years ');
         6:writeln('You are PORK year!   ',age,' years ');
         7:writeln('You are RAT year!   ',age,' years ');
         8:writeln('You are COW year!   ',age,' years ');
         9:writeln('You are TIGER year!,   ',age,' years ');
        10:writeln('You are RABBIT year!   ',age,' years ');
        11:writeln('You are DRAKON year!   ',age,' years ');
        end;
        write('enter birth year ');
        readln(your);
        age:=this-your
     end
end.
```

```
enter this year 3000 [enter]
enter birth year 2000 [enter]
You are Hen year! 1000 yeras
enter birth year 0
```

1 ให้ this เป็นปีปัจจุบัน

2 ให้ your เป็นปีเกิด

3 ดำเนินการคำนวณว่า ปีที่เกิดเป็นปีใดใน 12 ราศี จนกว่าจะใส่ข้อมูล 0 ให้ your จึงเลิกดำเนินการ

< คำสั่ง if..then..else..if >

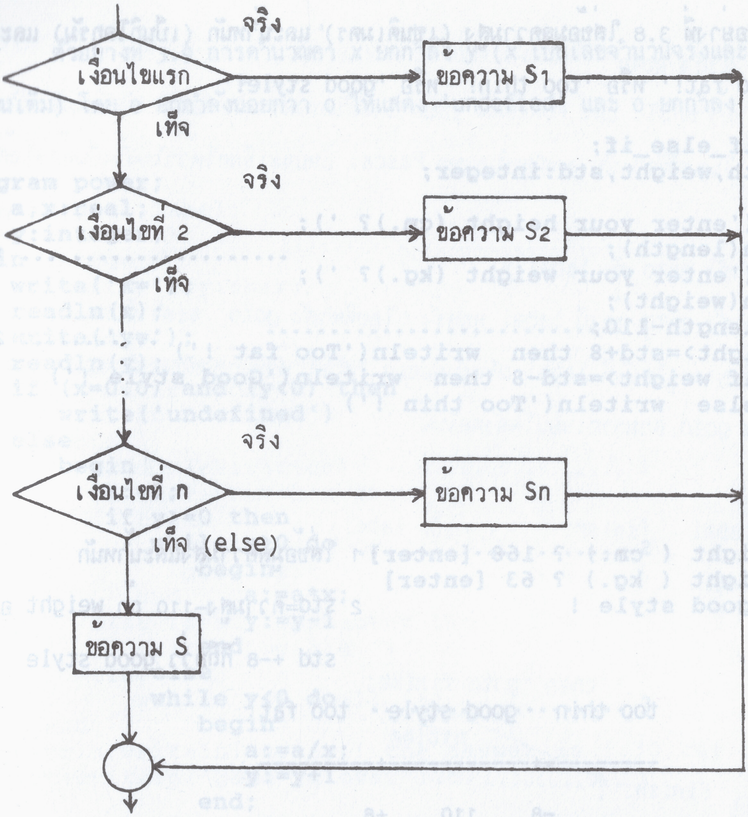
ในกรณีที่มีเพียงหนึ่งเงื่อนไข โดยปรกติจะใช้คำสั่ง if..then..else แต่ในกรณีที่มีมากกว่าหนึ่งเงื่อนไข นอกจากคำสั่ง case..of แล้ว ยังสามารถใช้คำสั่ง if..then..else ต่อหลัง if..then..else ได้ด้วย คำสั่ง case เป็นคำสั่งที่มีโครงสร้างที่เข้าใจง่าย แต่มีข้อเสียที่นิพจนหรือตัวแปรที่ใช้ในคำสั่ง case ไม่สามารถใช้ตัวแปรแบบเลขจำนวนจริงหรือสตริงได้ และยังไม่สามารถใช้การเปรียบเทียบอีกด้วย ดังนั้นจึงต้องคำนึงถึงความเหมาะสมว่าควรใช้คำสั่งใดด้วย การใช้คำสั่ง if หลังคำสั่ง if..then..else เรียกรวมกันเป็นคำสั่ง if..then..else..if

คำสั่ง if..then..else..if สามารถเขียนเป็นลักษณะทั่วไปได้ดังนี้คือ

```
if          เงื่อนไขแรก          then
    begin
        ข้อความ S
    end             (* ห้ามใส่ ; *)
else if      เงื่อนไขที่ 2        then
        begin
            ข้อความ S2
        end    (* ห้ามใส่ ; *)
                .

                .
        else if     เงื่อนไขที่ n     then
            then
                ข้อความ Sn
            end    (* ห้ามใส่ ; *)
        else
            begin
                ข้อความ S
            end;
```

ถ้าข้อความใดใน $S_1, S_2 \ldots S_n$ และ S มีเพียงหนึ่งข้อความ ก็ไม่จำเป็นต้องมีคำสั่ง begin .. end ครอบ



ตั้งแต่เงื่อนไขแรกจนถึงเงื่อนไขสุดท้าย ถ้ามีเงื่อนไขหนึ่งเป็นจริง ข้อความที่อยู่ตรงกับเงื่อนไขนั้นก็จะดำเนินการ ถ้าทุกเงื่อนไขเป็นเท็จหมดก็ดำเนินการข้อความ S หลังจากนั้นก็จะออกนอกคำสั่ง if..then..else ในกรณีที่ไม่ต้องใช้ข้อความ S ก็ไม่จำเป็นต้องเขียนคำสั่ง else

FLOW CHART ของคำสั่ง case จะมีลักษณะเป็นแนวตั้ง แต่คำสั่ง if..then..else จะมีลักษณะเป็นแนวนอนดังรูป สำหรับการดำเนินการนั้นเหมือนกันทุกประการ

```
ex   if point>80 then                 (* ถ้าคะแนนมากกว่า 80 *)
        writeln ('good')
     else if point>60 then            (* ถ้าคะแนนมากกว่า 60 *)
        writeln ('pass')
     else writeln ('fail')            (* ถ้าคะแนนต่ำกว่า 60 *)
```

เงื่อนไขใดที่เป็นจริง ก็จะดำเนินการข้อความหลัง then ของเงื่อนไขนั้น ถ้าเงื่อนไขเป็นเท็จทุกเงื่อนไข ก็จะดำเนินการข้อความหลัง else สำหรับเงื่อนไขที่ 2 นั้นไม่จำเป็นต้องเขียนเป็น (point<=80) and (point>60) เพราะผ่านเงื่อนไขแรกแล้ว

ตัวอย่างที่ 3.8 ใส่ข้อมูลความสูง (เซนติเมตร) และน้ำหนัก (เป็นกิโลกรัม) และแสดงผลมาว่า 'too fat!' หรือ 'too thin!' หรือ 'good style!

```
program if_else_if;
var length,weight,std:integer;
begin
  write('enter your height (cm.)? ');
  readln(length);                              ..................... 1
  write('enter your weight (kg.)? ');
  readln(weight);
  std:=length-110;
  if weight>=std+8 then  writeln('Too fat !') ........... 2
  else if weight>=std-8 then  writeln('Good style !')
       else  writeln('Too thin !')
end.
```

```
enter your height ( cm.) ? 168 [enter]
enter your weight ( kg.) ? 63 [enter]
You are very good style !
```

1 ใส่ข้อมูลความสูงและน้ำหนัก

2 std=ความสูง-110 ถ้า weight อยู่ใน std +-8 ก็ถือว่า good style

```
        too thin   good style   too fat
       ---------+--------------+---------
               -8     110     +8
```

< คำสั่ง if then else > (ต่อ)

ในกรณีที่ใช้ คำสั่ง if..then..else ต่อหลังคำสั่ง else สามารถเขียนเป็น FLOW-CHART และทำความเข้าใจได้ไม่ยากนัก แต่ในกรณีที่เป็นคำสั่ง if..then..else ต่อหลังคำสั่ง then อาจทำให้สับสนได้ง่ายดังตัวอย่างข้างล่างทางซ้าย

if เงื่อนไขแรก then	if เงื่อนไขแรก then
if เงื่อนไขที่ 2 then	begin
ข้อความ S_1	if เงื่อนไขที่ 2 then
else	ข้อความ S_1
ข้อความ S_2	else
	ข้อความ S_2
	end;

ในกรณีดังกล่าว else จะคู่กับ if ตัวสุดท้าย ทางที่ดีควรใช้คำสั่ง begin แบบขวา

ตัวอย่างที่ 3.9 การคำนวณค่า x ยกกำลัง y (x เป็นเลขจำนวนจริงและ y เป็นเลขจำนวนเต็ม) โดย 0 ยกกำลังน้อยกว่า 0 ให้แสดง 'undefied' และ 0 ยกกำลัง 0 เท่ากับ 1

```
program power;
var a,x:real;
    y:integer;
begin
    write('x=');
    readln(x);
    write('y=');          ........................... 1
    readln(y);
    if (x=0.0) and (y<0) then
       write('undefined')
    else
       begin
          a:=1;
          if y>=0 then
             while y>0 do  ..................... 2
                begin
                   a:=a*x;
                   y:=y-1
                end
             else
             while y<0 do  ..................... 3
                begin
                   a:=a/x;
                   y:=y+1
                end;
       writeln(a:10:2);     ...................... 4
       end;
end.
```

```
x=4 [enter]
y=10 [enter]
1048576.00
```

1 ใส่ข้อมูล x และ y

2 ถ้า y มีค่าเป็นบวก a จะมีค่า 1 x x ....(จำนวน y ครั้ง) ก็คือ x ยกกำลัง y

3 ถ้า y มีค่าเป็นลบ a จะมีค่า 1 1/x 1/x..(จำนวน y ครั้ง) ก็คือ x ยกกำลัง y

4 แสดงเอาท์พุท

3.7 คำสั่ง goto

ภาษา BASIC และ FORTRAN เป็นภาษาที่มีโครงสร้างไม่ทันสมัย ดังนั้นจึงจำเป็นต้องใช้คำสั่ง goto ในกรณีที่จำเป็น แต่สำหรับภาษา Pascal แทบจะเรียกได้ว่าไม่จำเป็นต้องใช้คำสั่ง goto เลย

< คำสั่ง goto กับ label >

คำสั่ง goto ต้องมี label ทุกครั้ง โดยที่คำสั่ง goto จะดำเนินการกระโดดจากตำแหน่งที่ดำเนินการอยู่ไปยังที่ label สำหรับชื่อของ label ต้องมีการกำหนดในตอนต้น ลักษณะทั่วไปของคำสั่ง goto สามารถเขียนได้ดังต่อไปนี้

```
label   finish;     ( ตั้งชื่อ label)
begin

   if  ...  then  goto finish;
                          กระโดด
   .
   finish: ;
end.
```

หลัง label ต้องมีเครื่องหมาย (:) เสมอ และหลัง label ต้องมีข้อความอยู่ด้วย ในกรณีที่ไม่จำเป็นเขียนข้อความใดๆ ก็ต้องมี (;) จากตัวอย่างข้างบนเปรียบเสมือนกับมีข้อความที่ว่างอยู่หลัง finish และก่อน end

ในกรณีของการใช้คำสั่ง end ก็มีข้อความว่างเช่นกัน ดังตัวอย่าง

```
ex   begin                                  begin
       a:=a+1;                                a:=a+1;
       b:=b-1     ←ไม่มี ;                    b:=b-1;
     end;                                   end;
```

จากตัวอย่างทั้งสอง โดยปรกติแล้วข้อความก่อน end ไม่ต้องมี (;) ถ้ามี (;) ก็หมายถึงก่อน end มีข้อความว่างอยู่

ใน Standard Pascal ชื่อของ label ต้องเป็นตัวเลขจำนวนเต็ม แต่ใน Turbo Pascal ชื่อของ label สามารถตั้งได้ตามกฎที่มีในบทที่ 1.3

ตัวอย่างที่ 3.9 โปรแกรมทายตัวอักษรอังกฤษตั้งแต่ a ถึง z โดยจะสุ่มตัวอักษรขึ้นมาตัวหนึ่ง (ใช้คำสั่ง random) และต้องทายให้ถูกภายใน 5 ครั้ง ถ้าหากมากกว่านั้นก็หมดสิทธิทาย

```
program goto_label;
label exit;     ........................... 1
var k:integer;
    character,key:char;
begin
   randomize;   ........................... 2
   key:=chr(random(26)+97); .............. 3
   write('enter from (a - z) ');
   for k:=1 to 5 do
      begin
        readln(character); ................ 4
        if character=key then
          begin
            writeln('good !':23);
            goto exit   ................... 5
          end
        else if key<character then
          write('too big ')
        else            ................... 6
          write('too small ');
      end;
   writeln;writeln('sorry ! the answer is ':30,key);
   exit:writeln('game is over':26) .......... 7
end.
```

```
enter from (a - z) m [enter]
too big g     [enter]
too big d     [enter]
too big b     [enter]
too small c   [enter]
                 good !
          game is over
```

1 กำหนดชื่อ label

2 คำสั่ง randomize เพื่อเริ่มการสุ่มตัวเลข

3 ใช้คำสั่ง random(26) เพื่อสุ่มค่าระหว่าง 0 ถึง 25 แล้วบวก 97 ผลจะได้เป็น 97 ถึง 122 ต่อจากนั้นใช้ฟังก์ชันchr(ดูตัวอย่างที่ 6) ก็จะได้ตัวอักษร a ถึง z ตัวใดตัวหนึ่ง

4 อ่านตัวอักษร a ถึง z

5 เมื่อค่าตอบถูกจะกระโดดไปที่ exit

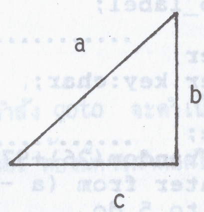
6 ถ้าตัวอักษรที่ใส่อยู่หลังค่าตอบจะแสดง 'too big' ถ้าอยู่หน้าก็ 'too small'

7 แสดง 'game is over'

ตัวอย่างที่ 3.11 ใส่ข้อมูลชนิดเลขจำนวนเต็ม โดยกำหนดให้ข้อมูลที่ใส่คือ ด้านยาวสุดของสามเหลี่ยมมุมฉาก และแสดงชนิดของสามเหลี่ยมมุมฉากทุกชนิดที่มีด้านที่ยาวที่สุดเท่ากับหรือสั้นกว่าข้อมูลที่ใส่เข้าไป

< วิธีคิด >

เงื่อนไขของสามเหลี่ยมมุมฉากคือ



$$a^2 = b^2 + c^2$$

ถ้า n เป็นข้อมูลที่ใส่เข้าไป

หลักการคิดก็คือ ให้ a,b,c เป็นตัวแปรลูปซ้อนลูปกัน 3 ลูป และตรวจดูเงื่อนไขว่า

$a^2 = b^2 + c^2$ เป็นจริงหรือไม่ ในกรณีที่เป็นจริง ให้แสดงค่า a,b และ c สำหรับเงื่อนไขการเขียนโปรแกรมมีดังนี้คือ

ด้าน a ยาวกว่าด้าน b และ c ดำเนินการวนซ้ำค่า a ตั้งแต่ 1 ถึง n

ด้าน c ยาวกว่าด้าน b (เพื่อไม่ให้แสดงสามเหลี่ยมชนิดเดียวกัน)

```
program triangle;
var a,b,c,n:integer;
begin
   write('what is the maximum value ? ');
   readln(n);            ................................. 1
   for a:=1 to n do
      for b:=1 to a do
         for c:=b to a do
            if a*a=b*b+c*c then
               writeln(a:8,b:8,c:8)  .......... 2
end.
```

```
what is the maximum value ? 18 [enter]
       5        3        4
      10        6        8
      13        5       12
      15        9       12
      17        8       15
```

1 ใส่ค่า n

2 แสดงค่า a,b,c ในกรณีที่เงื่อนไขเป็นจริง (เป็นสามเหลี่ยมมุมฉาก)

ตัวอย่างที่ 3.12 โปรแกรมหาคำตอบของสมการ $ax^2 + bx + c = 0$ โดยใส่ค่าตัวแปร a,b และ c และแสดงผลออกมาที่จอ

< วิธีคิด > จากสมการข้างบน เมื่อใช้สูตร

$$y = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$ ก็จะได้ y เป็นคำตอบ

ก่อนอื่นต้องทำการคำนวณหาค่า b*b-4*a*c ก่อน โดยให้ค่าดังกล่าวเท่ากับ d ต่อไปคือการหารากที่สองซึ่งใช้ฟังก์ชัน sqrt โดยให้ sd มีค่าเป็นรากที่สองของ d ข้อควรระวังสำหรับปัญหานี้คือค่า d อาจจะน้อยกว่า 0 จึงต้องใช้ฟังก์ชัน abs ซึ่งเป็นการกำหนดค่าสัมบูรณ์ของ d หลังจากนั้นก็ทำการแก้ปัญหานี้ได้โดยแยก sd เป็น 3 กรณีคือ d>0 ,d=0 และ d<0

```
program quadratic_equation;
var a,b,c:integer;
    d,sd:real;
begin
  readln(a,b,c); ................................ 1
  writeln('a=',a,' b=',b,' c=',c); ............ 2
  d:=b*b-4*a*c;
  sd:=sqrt(abs(d)); ............................ 3
  if d>0 then
     writeln('the answer =',-b/(2*a)+sd/(2*a):7:3,
     ' , ',d-b/(2*a)-sd/(2*a):7:3)
   else if d=0 then
     writeln('the answer =',-b/(2*a):7:3).... 4
  else
     writeln('the answer =',-b/(2*a):7:3,
     ' + -',abs(sd/(2*a)):7:3,'i')
  end.
```

```
2 3 5 [enter]
a=2 b=3 c=5
the answer = -0.750 + -  1.392i
```

1 ใส่ค่า a,b,c

2 แสดงค่า a,b,c

3 หาค่าของ d และ sd(รากที่สองของ d)

4 แสดงคำตอบโดยแยกเป็น 3 กรณี คือ sd > 0 , sd = 0 และ sd < 0

ตัวอย่างที่ 3.13 โปรแกรมเครื่องคิดเลขอย่างง่าย ซึ่งสามารถทำการคำนวณ +-*/ แต่ทำการคำนวณเลขได้เพียงหนึ่งคู่ โดยใส่อินพุทดังตัวอย่าง

20 _+_ 10 [enter]

< วิธีคิด > ก่อนอื่นต้องกำหนดโอเปอเรเตอร์ทั้งสี่ตัวคือ + (บวก),- (ลบ), * (คูณ) และ / (หาร) เมื่อกำหนดโอเปอร์เรเตอร์เรียบร้อยแล้ว ก็ใช้คำสั่ง case..of

```
program calculate;
var a,b:real;
  ope,dumy:char;
begin
  readln(a,dumy,ope,b); ......................... 1
  write('the answer is =');
        case ope of   ......................... 2
            '+':writeln(a+b:10:2);
            '-':writeln(a-b:10:2);
            '*':writeln(a*b:10:2);
            '/':writeln(a/b:10:2);
            else writeln('operator error')
        end
end.
```

```
  4 * 60 [enter]
  the answer is =    240.00
```

1 ใส่อินพุท 4 _ * _ 60 [enter] จากคำสั่ง read(a,dumy,ope,b) a คือ 4 dumy คือ _ ope คือ * และ b คือ 60 ถ้าไม่มีตัวแปร dumy ope จะเป็นตัวเว้นวรรค หรือ space

2 คำนวณ + - * / ตามโอเปอร์เรเตอร์ที่อ่านค่าได้

ตัวอย่างที่ 3.14 โปรแกรมเครื่องคิดเลข (แบบที่สอง)

< วิธีคิด > จากตัวอย่างที่แล้วการคำนวณสามารถคำนวณได้เพียงเลขสองตัว ในกรณีที่ต้องการบวกลบคราวละมากกว่าสองตัวสามารถทำได้ดังนี้

อ่านค่าข้อมูลที่ใส่เข้ามาโดยวิธี read(kbd,ข้อมูล) วิธีการอ่านแบบนี้เป็นการอ่านข้อมูลทางคีย์บอร์ด โดยคอมพิวเตอร์จะรับข้อมูลทันทีที่กดคีย์บอร์ด การอ่านข้อมูลแบบนี้ทำให้ไม่จำเป็นต้องกด enter ทุกครั้ง ซึ่งสะดวกกว่า แต่ต้องใช้คำสั่ง uses turbo3 เรียก unit turbo3 มาใช้

```
program calculater;
uses turbo3;
const
   plus = '+';
   minus ='-';
   term =';';
var
   number,result:integer; ............................ 1
   sign,ch:char;
begin
result:=0;
sign:='+';
while (sign<>term) do        ......................... 2
   begin
      number := 0;
      read(kbd,ch); write(ch);
      while(('0'<=ch) and (ch<='9')) do
         begin
            number:=10*number+ord(ch)-ord('0'); .... 3
            read(kbd,ch); write(ch);
         end;
      if sign = plus
         then result := result+number
            else if sign = minus        ................ 4
               then result := result-number;
      sign := ch
   end;
   writeln;
writeln(result)
end.
```

```
      456-123+789;
      1122
```

1 กำหนดให้ result เป็นตัวแปรคำตอบและ number เป็นตัวแปรของข้อมูลที่ใส่เข้าไป

2 คำสั่งวนซ้ำจนกว่า sign เท่ากับ ;

3 ดำเนินการคำนวณเลขมากกว่า 1 หลัก เช่น 28 -> 10*2+ord('8')-ord('0')

4 คำนวณตามค่า + และ -

แบบฝึกหัด

1 จงเขียนโปรแกรมหาผลบวกของ 1 ถึง n โดยให้ n เป็นค่าอินพุทที่สามารถกำหนดได้ตามความต้องการ และห้ามใช้สูตร n(n+1)/2

2 จากโปรแกรมข้างล่าง จงหา

```
program what_is_done;
var i,d,s:integer;
begin
   i:=0;d:=1;s:=0;
   while i<5 do
     begin
       i:=i+1;
       s:=s+d;
       d:=d+2;
       writeln(i:5,d:5,:5)
     end
end.
```

ก) การเปลี่ยนแปลงของ i,d,s

ข) เมื่อสิ้นสุดการดำเนินการแล้ว จงหาความสัมพันธ์ของ i กับ s

ค) จงพิสูจน์ข้อ ข)

3 x mod y = 0 หมายความว่า x หาร y ได้ลงตัว จงใช้โอเปอร์เรเตอร์ mod แยกตัวประกอบเลขจำนวนเต็มตัวหนึ่ง (ใส่อินพุทเอง)

4 จงเขียนโปรแกรมรับข้อมูล 10 ข้อมูล และแสดงผลของข้อมูลที่มีค่า มากที่สุด มากที่สอง น้อยที่สุด และน้อยที่สอง

5 จงหาเอาท์พุทของโปรแกรมข้างล่างในกรณีที่อินพุทคือ 0 และ 1

```
program test_you;
var a,b,c:integer;
begin
  a:=1;
  b:=1;
  readln(c);
  if c=1 then
     a:=2;
   else
     b:=2; c:=2;
  writeln(a:3,b:3,c:3)
 end.
```

6 จงเขียนโปรแกรมหาจำนวนเต่ากับนก โดยใส่ข้อมูลของจำนวนรวมของเต่ากับนก และจำนวนรวมของขาเต่ากับขานก

บทที่ 4

อาร์เรย์และสตริง

4.1 ข้อมูลแบบอาร์เรย์ (array)

สมมุติว่ามีข้อมูลอยู่ 100 ข้อมูล ถ้าหากว่าต้องการใช้ตัวแปรเพื่อเก็บค่าของข้อมูลชุดนี้ ก็จำเป็นจะต้องประกาศตัวแปรโดยวิธี

var a1,a2,... ไปจนถึง a100:integer;

จะเห็นได้ว่าเป็นการเสียเวลาอย่างมาก ในกรณีที่ใช้ตัวแปรแบบอาร์เรย์จะสะดวกกว่า เนื่องจากตัวแปรแบบนี้มีอินเด็กซ์เป็นตัวชี้ว่าตัวแปรที่เท่าใด ถ้า a เป็นตัวแปรแบบอาร์เรย์ ตัวแปรแรกคือ a[1] ตัวที่สองก็คือ a[2] .... ตัวสุดท้ายก็คือ a[100] ตัวเลขในวงเล็บก้ามปู ([]) หมายถึง ลำดับของข้อมูล (เรียกว่า อินเด็กซ์) ข้อมูลแบบนี้สะดวกกว่าข้อมูลแบบ a1,a2 .. a100 เพราะภายในวงเล็บก้ามปู สามารถใช้ตัวแปรได้ เมื่อต้องการใช้ข้อมูลใดก็เพียงอ้างค่าอินเด็กซ์ใน [] ก็สามารถนำข้อมูลนั้นมาใช้ได้ แต่ตัวแปรแบบ a1,a2..a100 การเรียกข้อมูลไม่สามารถอ้างค่า 1..100 ได้ เนื่องจาก 1..100 เป็นส่วนหนึ่งของตัวแปร

สมมุติว่าต้องการให้ข้อมูลทั้ง 100 ข้อมูลมีค่าเป็น 0 สามารถเขียนได้ดังตัวอย่าง

```
for i:=1 to 100 do
  a[i]:=0;
```

< การประกาศข้อมูลแบบอาร์เรย์ >

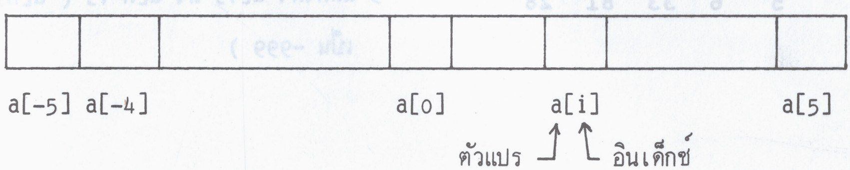
การประกาศข้อมูลแบบอาร์เรย์ สามารถประกาศได้ดังนี้

var ชื่อตัวแปร,ชื่อตัวแปร..:array [อินเด็กซ์แรก..อินเด็กซ์สุดท้าย] of แบบข้อมูล;

วิธีเขียนอินเด็กซ์ต้องเขียนอินเด็กซ์แรกตามด้วย .. (พีเรียดสองตัว) และอินเด็กซ์ตัวสุดท้าย อินเด็กซ์ในภาษา Pascal นั้นยืดหยุ่นได้ กล่าวคือ จะใช้เลขจำนวนเต็มที่มีค่าเป็นลบหรือตัวอักขระก็ได้ เช่น -5..10 หรือ 'a'..'z' แบบข้อมูลจะเป็น real,integer,charหรือข้อมูลที่ซับซ้อนซึ่งจะกล่าวในบทต่อไปก็ได้

ex var a:array[-5..5] of integer;

หมายถึง a เป็นตัวแปรแบบอาร์เรย์มีค่าเป็นเลขจำนวนเต็ม ซึ่งมีอินเด็กซ์ตั้งแต่ -5 ถึง 5 กล่าวคือ มีตัวแปร a[-5]..a[0]..a[-5] รวม 11 ตัว ตัวแปรตัวที่ i สามารถเขียนเป็น a[i] ได้



การให้ค่าแก่ตัวแปรแบบนี้สามารถให้ค่าได้เหมือนกับตัวแปรธรรมดา ในกรณีที่ต้องการส่งค่าทุกค่าจากตัวแปรแบบอาร์เรย์ ให้กับตัวแปรแบบเดียวกันสามารถเขียนได้ดังนี้

ถ้าประกาศให้ var a,b:array [1..100] of integer;

a:=b; หมายถึง ตัวแปร a[ ] ที่มีอินเด็กซ์ตรงกับตัวแปร b[ ] จะมีค่าเท่ากัน

ex var a,b:array [-5..10] of integer;

c:array ['a'..'z'] of integer;

c['a']	c['b']	....	c['z']

ตัวอย่างที่ 4.1 ใส่ข้อมูลเลขจำนวนเต็มให้กับตัวแปรแบบอาร์เรย์ เมื่อสิ้นสุดการใส่ข้อมูลให้แสดงข้อมูลที่ใส่เข้าไปทุกค่าออกมา

```
program array_1;
var a:array[1..50] of integer;   ..................... 1
    j,n:integer;
begin
   n:=1;
   readln(a[n]);
   while a[n]<>-999 do
      begin
         n:=n+1;
         readln(a[n])             ..................... 2
      end;
   for j:=1 to n-1 do             ..................... 3
      write(a[j]:4);
   writeln
end.
```

```
6      [enter]
33     [enter]
81     [enter]
28     [enter]
-999   [enter]

   5   6   33   81   28
```

1 ประกาศตัวแปรแบบอาร์เรย์ a[1]..a[50]

2 ใส่ข้อมูล a[n] และเพิ่มค่า n ทุกครั้งหลังการใส่ข้อมูล

3 แสดงค่า a[1] ถึง a[n-1] (a[n] มีค่าเป็น -999)

ตัวอย่างที่ 4.2 ใส่ข้อมูลตัวเลขอย่างละกี่ตัวก็ได้ และนับว่าข้อมูลที่ใส่มีเลข 0 ถึง 9 อย่างละกี่ตัว ถ้าข้อมูลที่ใส่เป็น '/' ก็เป็นอันสิ้นสุดการใส่ข้อมูล และแสดงผลว่ามีอย่างละกี่ตัวโดยใช้ตัวแปรแบบอาร์เรย์ ซึ่งใช้ตัวอักขระเป็นอินเด็กซ์ (a['0'] .. a['9'])

```
program distribute;
var count:array['0'..'9'] of integer;   ................ 1
  c,i:char;
begin
   for i:='0' to '9' do
      count[i]:=0;
   readln(c);
   while c<> '/' do
      begin
         if ('0'<=c) and (c<='9') then
            count[c]:=count[c]+1;         ................ 2
         readln(c)
      end;
   for i:='0' to '9' do
      writeln(i:2,' :',count[i]:4)
end.
```

1 ประกาศตัวแปรแบบอาร์เรย์ count['0'] ถึง count['9']

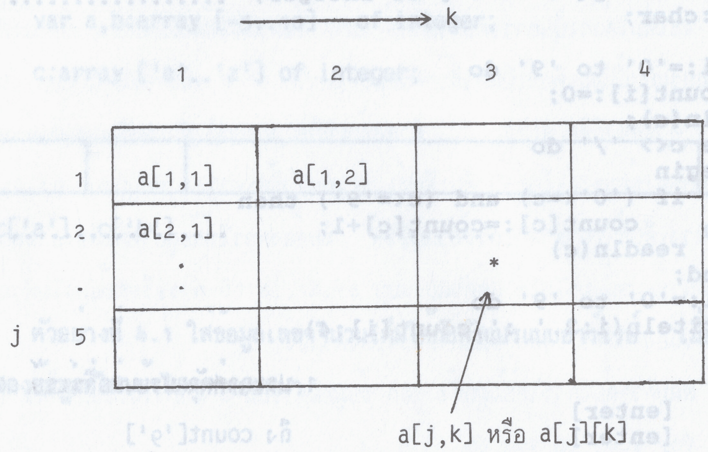
2 ถ้าข้อมูล c ที่ใส่เข้ามาอยู่ระหว่าง 0 ถึง 9 ให้เพิ่มค่า count[c] ที่มีอินเด็กซ์ตรงกับตัวเลขที่ใส่เข้าอีก 1

```
    5   [enter]
    6   [enter]
    7   [enter]
    8   [enter]
    9   [enter]
    5   [enter]
    5   [enter]
    /   [enter]
  0 :    0
  1 :    0
  2 :    0
  3 :    0
  4 :    0
  5 :    3
  6 :    1
  7 :    1
  8 :    1
  9 :    1
```

< อาร์เรย์หลายมิติ >

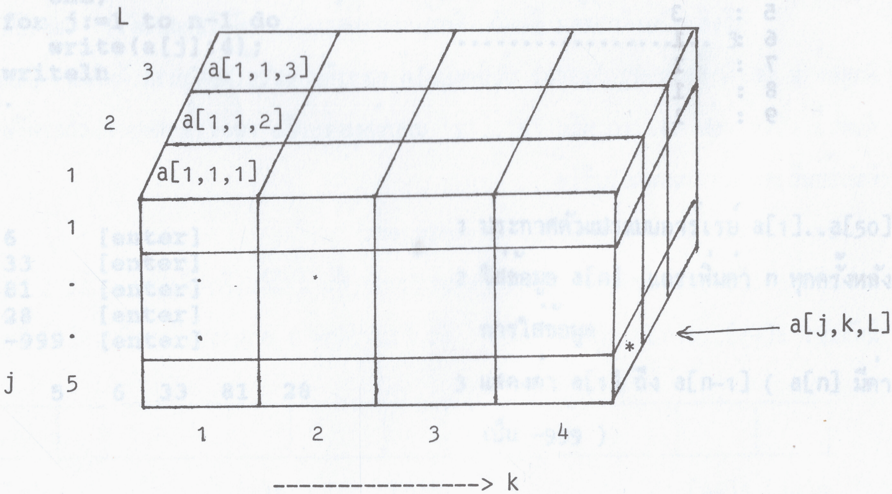
อาร์เรย์แบบที่กล่าวในตอนต้นเป็นการจัดลำดับแบบเส้นแนวนอน ข้อมูลแบบนี้ยังสามารถขยายเป็น อาร์เรย์สองมิติได้อีกด้วย วิธีการประกาศมีดังนี้คือ

```
var a:array[1..5,1..4] of integer;
```



อาร์เรย์สองมิติมีลักษณะเป็นระนาบโดยมีตัวแปร j,k เป็นอินเด็กซ์ อาร์เรย์สามมิติก็สามารถสร้างได้โดยหลักการเดียวกันเป็นดังที่เห็น

```
var a:array[1..5,1..4,1..3] of integer;
```



ตัวอย่างที่ 4.3 ใส่คะแนนวิชาภาษาอังกฤษ เลข และวิทยาศาสตร์ของนักเรียน 3 คน และแสดงผลเมื่อใส่ข้อมูลเสร็จ

```
program markpoint;
const subjectnum=3;
      studentnum=3;
var subjectject:array[1..subjectnum] of char; ................ 1
    point:array[1..subjectnum,1..studentnum] of integer; ... 2
    i,j:integer;
begin
   subjectject[1]:='E';
   subjectject[2]:='M';   ........................................ 3
   subjectject[3]:='S';
   for j:=1 to subjectnum do
      begin
         writeln(subjectject[j]);
         for i:=1 to studentnum do
            begin
               write('STUDENT NUMBER',i:3,' ? ');
               readln(point[i,j]); ............................ 4
            end
      end;
  writeln(' ':25,'ENGLISH':10,'SCIENCE':15,'MATHEMATICS':18);
  for i:=1 to studentnum do
     begin
        write('STUDENT NUMBER',i:3);
        for j:=1 to subjectnum do
           write(point[i,j]:15); ............................... 5
        writeln
     end
end.
```

1 subject [] เป็นตัวแปรเก็บตัวอักขระ

2 point[,] เป็น อาร์เรย์ 2 มิติเพื่อเก็บคะแนน

3 ให้ค่าแก่ subject [1],[2],[3]เป็น 'E', 'M', 'S' ตามลำดับ

4 ให้ค่าคะแนนของนักเรียนให้แก่ตัวแปร point[,]

5 แสดงคะแนนของนักเรียนทุกคน

```
E
STUDENT NUMBER   1 ?  55   [enter]
STUDENT NUMBER   2 ?  58   [enter]
STUDENT NUMBER   3 ?  60   [enter]
M
STUDENT NUMBER   1 ?  65   [enter]
STUDENT NUMBER   2 ?  76   [enter]
STUDENT NUMBER   3 ?  75   [enter]
S
STUDENT NUMBER   1 ?  78   [enter]
STUDENT NUMBER   2 ?  70   [enter]
STUDENT NUMBER   3 ?  68   [enter]
```

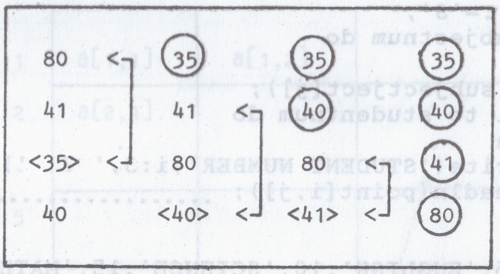
```
                         ENGLISH          SCIENCE        MATHEMATICS
STUDENT NUMBER  1          55               65               78
STUDENT NUMBER  2          58               76               70
STUDENT NUMBER  3          60               75               68
```

ตัวอย่างที่ 4.4 ใส่ข้อมูลแบบเลขจำนวนเต็มชุดหนึ่งและเรียงลำดับจากน้อยไปหามาก

< วิธีคิด > การเรียงลำดับนั้นมีอยู่หลายวิธีด้วยกัน สำหรับปัญหานี้จะใช้การเรียงแบบ SIMPLE SORT ซึ่งมีหลักการคิดดังนี้

สมมุติว่ามีข้อมูล n ข้อมูล ให้เลือกเอาข้อมูลที่มีค่าน้อยที่สุดมา และสลับตำแหน่งกับข้อมูลแรก ข้อมูลที่เหลืออีก n-1 ข้อมูล ก็ใช้วิธีเดียวกัน

รูปข้างล่างเป็นการแสดงการดำเนินการของข้อมูล 5 ข้อมูล



< > ข้อมูลที่มีค่าน้อยที่สุด , <——> สลับตำแหน่ง , ◯ ข้อมูลที่จัดเรียบร้อยแล้ว

```
program sort;
const n=4;
var a:array[1..100] of integer;
   k,L,min,s,dumy:integer;
begin
  a[1]:=80; a[2]:=41; a[3]:=35; a[4]:=40; .................... 1
  for k:=1 to n-1 do
    begin
      min:=a[k]; s:=k; ................................ 2
      for L:=k+1 to n do
        if a[L]<min then
          begin
            min:=a[L]; s:=L; .......................... 3
          end;
      dumy:=a[k]; a[k]:=a[s]; a[s]:=dumy .................. 4
    end;
  for k:=1 to n do
    write(a[k]:5);
  writeln
end.
```

35   40   41   80

1 ค่าของ a[1] ถึง a[4]

2 ให้ข้อมูลตัวที่ k มีค่าน้อยที่สุด

3 ตรวจดูข้อมูลหลังจากตัวที่ k คือ ตั้งแต่ตัวที่ k+1 จนถึงตัวสุดท้ายว่าตัวใดมีค่าน้อยที่สุด โดยให้ min เท่ากับค่าที่น้อยที่สุด และ s เป็นค่าของลำดับของข้อมูลที่น้อยที่สุด

4 สลับค่า k กับ s (สลับตำแหน่งของค่าน้อยที่สุดไปที่ k)

ตัวอย่างที่ 4.5 ใส่ข้อมูลระหว่าง 0 ถึง 9 และใช้ฮีสโตแกรมแสดงจำนวนข้อมูลที่ใส่ว่ามีอย่างละกี่ตัวดังตัวอย่าง

```
0 : *
1 : *****
2 : **......*
     count[2]
   .
9 : ***
```

< วิธีคิด > ให้ตัวแปร count เป็นอาร์เรย์ที่มีอินเด็กซ์เป็นตัวอักขระ '0' ถึง '9' เมื่ออ่านได้ตัวอักขระใดก็เพิ่มค่า count ที่มีอินเด็กซ์ตรงกับตัวอักขระนั้น และเมื่อสิ้นสุดการให้ข้อมูลให้แสดง '*' เป็นจำนวนเท่ากับค่าของ count [i] โดยใช้คำสั่ง

```
for k:=1 to count[i] do
   write('*');
```

```
program checknum;
var count:array['0'..'9'] of integer;
   c,i:char;
   k:integer;
begin
  for i:='0' to '9' do
    count[i]:=0;
  readln(c);
  while c<>'/' do
    begin
      if('0'<=c) and (c<='9') then
         count[c]:=count[c]+1;
      readln(c);
    end;
  for i:='0' to '9' do
    begin
      write(i,': ');          ............................ 1
      for k:=1 to count[i] do
      write('*');             ............................ 2
      writeln                 ..................... ...... 3
    end
end.
```

```
   4
   0
   /
   0: *
   1:
   2:
   3:
   4: *
   .
   9:
```

1 แสดง 0: ถึง 9:

2 แสดง '*' เท่ากับค่าของ count[i]

3 ขึ้นบรรทัดใหม่

(ในหนังสือแสดงเอาท์พุทเพียงบางส่วน)

ตัวอย่างที่ 4.6 ใส่ข้อมูล $X_1, X_2 .....X_n$ และคำนวณหาค่าเฉลี่ย ค่า VARIANCE และ STANDARD DEVIATION ของข้อมูลชุดนี้

< วิธีคิด > ค่า VARIANCE และ STANDARD DEVIATION สามารถหาได้จากสูตร

VARIANCE = ผลบวกของ Xi ยกกำลังสอง / จำนวนข้อมูล - ค่าเฉลี่ยยกกำลังสอง

STANDARD DEVIATION = รากที่สองของ VARIANCE (ใช้ฟังก์ชัน sqrt)

```
program.statistics;
var x:array[1..100] of real;
    s,m,t,v,sd:real;
    i,n:integer;
begin
  i:=1;
  readln(x[i]);
  while x[i]<>-999 do ...................................... 1
    begin
      i:=i+1;
      readln(x[i])
    end;
  s:=0;
  n:=i-1;
  for i:=1 to n do
    s:=s+x[i];
  m:=s/n;                        ........................... 2
  t:=0;
  for i:=1 to n do
    t:=t+sqr(x[i]);              ........................... 3
  v:=t/n-sqr(m);                 ........................... 4
  sd:=sqrt(v);                   ........................... 5
  writeln('mean                 =',m:7:3);
  writeln('variance             =',v:7:3);  ................6
  writeln('standard deviation =',sd:7:3)
end.
```

```
75    [enter]
61    [enter]
90    [enter]
56    [enter]
82    [enter]
-999  [enter]
mean                 = 72.800
variance             =161.360
standard deviation = 12.703
```

1 ใส่ข้อมูลจนกว่า a[] = -999

2 คำนวณหาค่าเฉลี่ย

3 คำนวณหาผลรวมของ Xi ยกกำลัง 2

4 คำนวณหาค่า v (VARIANCE)

5 ค่า sd(STANDARD DEVIATION)

6 แสดงผล

4.2 ข้อมูลแบบสตริง (string)

ใน Standard Pascal ไม่มีข้อมูลแบบสตริง แต่ใน Turbo Pascal มีข้อมูลแบบนี้ ข้อมูลแบบสตริงคือ ข้อมูลที่สามารถเก็บข้อมูลตัวอักขระได้หลายๆตัว เช่น ชื่อคน ชื่อสถานที่ เป็นต้น

<การประกาศข้อมูลแบบสตริง >

การประกาศข้อมูลแบบสตริงสามารถประกาศได้ดังนี้

var ตัวแปร,ตัวแปร ... : string[ความยาว];

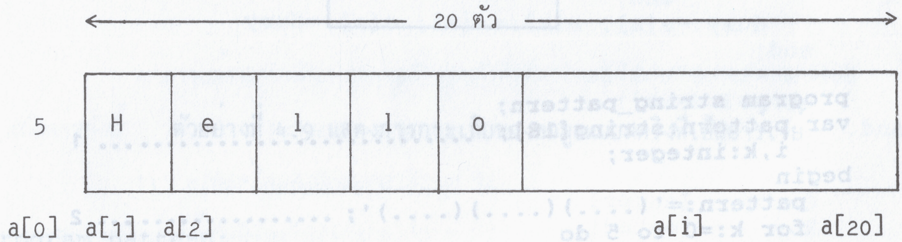
ความยาวของข้อมูลแบบนี้สามารถกำหนดได้ตั้งแต่ 1 ถึง 255 เมื่อประกาศ

var a:string[20];

ความหมายก็คือ ตัวแปรแบบสตริง a สามารถเก็บอักขระได้อย่างมาก 20 ตัว การให้ค่าแก่ตัวแปรแบบนี้สามารถเขียนได้ดังนี้

a:='Hello';

โครงสร้างของตัวแปร a จะเป็นดังที่เห็นข้างล่าง



อินเด็กซ์ของ a จะชี้ตัวอักขระที่ตำแหน่งนั้น จากตัวอย่างข้างบน a[1] คือ 'H' จนถึงตัวสุดท้าย a[20] (ในกรณีนี้ a[20] ไม่มีข้อมูลอยู่) ส่วน a[0] คือ ความยาวของข้อมูลที่อยู่ใน a นั่นคือ 5

< การรวมและการเปรียบเทียบ >

ข้อมูลแบบสตริงสามารถที่จะนำมารวมกันได้โดยใช้เครื่องหมาย (+) ดังตัวอย่าง

a:='Hello'+'_Pascal'

จากตัวอย่างข้างบน a จะมีค่าเป็น 'Hello_Pascal'

นอกจากนี้ ยังสามารถใช้โอเปอร์เรเตอร์เปรียบเทียบ =,<>,>,>= ฯลฯ ได้อีกด้วย โดยการเปรียบเทียบจะเป็นการเรียงตามลำดับอักขระ (เหมือนพจนานุกรม) ดังตัวอย่าง

'a'<''aa'<'ab'<'b'

ตัวอย่างที่ 4.7 ใส่ข้อมูลแบบสตริง และแสดงผลเป็นประโยค

```
program message;
var name:string[20]; ...................... 1
    year:integer;
begin
   write('name ? ');
   readln(name);
   write('year ? ');
   readln(year);
   writeln('Your name is ',name,'.'); .. 2
   writeln('I am ',year,' years old.');
end.
```

```
name ? Ann [enter]
year ? 23  [enter]
Your name is Ann.
I am 23 years old.
```

1 ประกาศตัวแปรสตริงเก็บอักขระ 20 ตัว

2 แสดงเอาท์พุทเป็นประโยค

ตัวอย่างที่ 4.8 แสดงการระเถิบไปทางซ้ายทีละตัวของข้อมูลแบบสตริง

```
program string_pattern;
var pattern:string[18]; ........................... 1
    i,k:integer;
begin
   pattern:='(....)(....)(....)'; ................. 2
   for k:=0 to 5 do
      begin
         for i:=(k mod 6)+1 to (k mod 6)+12 do .. 3
            write(pattern[i]);
         writeln
      end
end.
```

```
(....)(....)
....)(....)(
...)(....)(.
..)(....)(..
.)(....)(...
)(....)(....
```

1 ประกาศตัวแปรแบบสตริง

2 ให้ pattern เป็น '(....)(....)(....)'

3 ระหว่างที่ k เปลี่ยนจาก 0 ถึง 5 ค่าของ i จะเปลี่ยนเป็น 1,2..12, 2,3..13

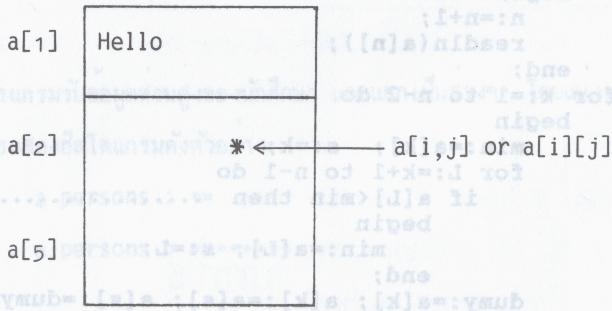
< การใช้อาร์เรย์กับข้อมูลแบบสตริง >

ข้อมูลแบบสตริงเมื่อต้องการใช้อาร์เรย์ก็สามารถประกาศได้ดังต่อไปนี้

var a:array[1..5] of string[10];

จากการประกาศตัวแปร a ข้างบน a[1] ถึง a[5] จะเป็นตัวแปรแบบสตริงที่สามารถเก็บตัวอักขระได้ 10 ตัว

การให้ค่าก็สามารถให้ค่าได้เหมือนตัวแปรแบบสตริงทั่วไป เช่น a[1]:='Hello' และในกรณีที่ต้องการเรียกตัวอักขระเพียงตัวเดียวก็สามารถเรียกโดยเรียก a[i,j] หรือ a[i][j]



ตัวอย่างที่ 4.9 แสดงการกระเก็บของข้อมูลแบบสตริงที่เป็นอาร์เรย์

```
program pattern;
var sentence:array[1..3] of string[10];  .................. 1
    j,k:integer;
begin
  sentence[1]:='[--------]';
  sentence[2]:='[ PASCAL ]';                .................. 2
  sentence[3]:='[--------]';
  for j:=0 to 5 do
    begin
      for k:=1 to 3 do
        write(sentence[j mod 3 +1]);      .................. 3
      writeln
    end
end.
```

```
[--------][--------][--------]
[ PASCAL ][ PASCAL ][ PASCAL ]
[--------][--------][--------]
[--------][--------][--------]
[ PASCAL ][ PASCAL ][ PASCAL ]
[--------][--------][--------]
```

1 ประกาศ sentence เป็นข้อมูลแบบสตริง

2 ให้ค่าแกสตริง sentence

3 ระหว่างที่ค่า j เปลี่ยนจาก 0 ถึง 5 นั้น j mod 3 +1 จะเปลี่ยนเป็น 1,2,3,1,2,3

ตัวอย่างที่ 4.10 การเรียงลำดับข้อมูลแบบสตริง

< วิธีคิด > หลักการคิดเหมือนกับการจัดลำดับตัวเลข โดยใช้หลักการที่ว่าการเปรียบเทียบข้อมูลแบบสตริงสามารถเปรียบเทียบในลักษณะ 'a'<'aa'<'ab'<'b' ได้

```
program sortname;
var a:array[1..100] of string[20];
    min,dumy:string[20];
    k,L,n,s:integer;
begin
   n:=1;
   read(a[n]);
   while a[n]<>'/' do .......................... 1
     begin
       n:=n+1;
       readln(a[n]);
     end;
   for k:=1 to n-2 do
     begin
       min:=a[k];   s:=k;
       for L:=k+1 to n-1 do
         if a[L]<min then  ................. 2
           begin
             min:=a[L]; s:=L;
           end;
       dumy:=a[k]; a[k]:=a[s]; a[s]:=dumy
     end;
   for k:=1 to n-1 do
     writeln(a[k])
end.
```

```
Candy    [enter]
Emy      [enter]
Nancy    [enter]
Rolla    [enter]
Madonna  [enter]
/

Candy
Emy
Madonna
Nancy
Rolla
```

1 อ่านชื่อจนกว่าจะใส่ '/'

2 การเปรียบเทียบข้อมูลแบบสตริงสามารถเปรียบเทียบตามปรกติ เช่น 'aa'<'zz'

แบบฝึกหัด

1 คำนวณและแสดงค่าของ 2 ยกกำลัง 200 การคำนวณดังกล่าวไม่สามารถใช้ตัวแปรที่มีอยู่ ดังนั้นจึงต้องมีการใช้ข้อมูลแบบอาร์เรย์โดยให้แต่ละตัวเป็นค่าแต่ละหลักของผลที่คำนวณได้

2 สามเหลี่ยมที่เห็น เรียกว่าสามเหลี่ยม Pascal จงเขียนสามเหลี่ยมนี้ขึ้นมา 10 แถว โดยมีหลักว่า ด้านข้างมีค่าเป็นหนึ่ง ส่วนค่าภายในเท่ากับผลบวกของค่าด้านบนซ้ายบวกด้านบนขวา

```
      1
     1 1
    1 2 1
   1 3 3 1
```

3 เขียนโปรแกรมรับข้อมูลส่วนสูงของนักศึกษา และแยกเป็นช่วงๆ โดยแบ่งเป็นช่วงละ 5 เซนติเมตร และทำการแสดงฮีสโตแกรมดังตัวอย่าง

```
150..155    2 persons : **
155..160    8 persons : ********
```

4 การแก้สมการตัวแปรหลายตัวแปรดังที่เห็น โดยวิธี GJ ASS ELIMINATION ซึ่งมีหลักการดังนี้

$$A_{11}X_1+A_{12}X_2+A_{13}X_3 = B_1$$

$$A_{21}X_1+A_{22}X_2+A_{23}X_3 = B_2$$

$$A_{31}X_1+A_{32}X_2+A_{33}X_3 = B_3$$

1) นำสมการแถวแรกคูณ (A_{21}/A_{11}) และนำไปลบจากแถวที่ 2 ทำให้ A_{21} เป็น 0 ต่อไปคูณแถวแรกด้วย (A_{31}/A_{11}) และนำไปลบจากแถวที่ 3 ซึ่ง A_{31} จะเป็น 0 ค่าสัมประสิทธิ์ของแถวแรกจะเปลี่ยนไป

2) ดำเนินการวิธีเดียวกับข้อ 1 โดยใช้แถวที่ 2 คูณ (A_{32}/A_{22}) และลบแถวที่ 3

3) ดำเนินการวิธีเดียวกับข้อ 1 โดยใช้แถวที่ 3 ทำให้ A_{13} และ A_{23} เป็น 0

4) ดำเนินการวิธีเดียวกับข้อ 2 โดยใช้แถวที่ 2 ทำให้ A_{12} เป็น 0

ผลที่ได้คือ $A_{11}'X_1 = B_1'$

$$A_{22}'X_2 = B_2'$$

$$A_{33}'X_3 = B_3'$$

จากผลข้างบนทำให้สามารถ หาค่า X_1 X_2 และ X_3 ได้ จงเขียนโปรแกรมแก้สมการหลายตัวแปรโดยสามารถเลือกว่าต้องการแก้สมการกี่ตัวแปร

บทที่ 5

แบบข้อมูลชนิดต่าง ๆ

5.1 ข้อมูลในภาษา Pascal

< ประเภทของข้อมูล >

ในบทก่อนๆได้กล่าวถึงข้อมูลแบบอักขระ เลขจำนวนเต็ม เลขจำนวนจริง และอาร์เรย์แล้ว ข้อมูลดังกล่าวเป็นข้อมูลที่อยู่ในภาษาคอมพิวเตอร์อื่นๆเช่นกัน

ในภาษา Pascal นอกจากข้อมูลที่กล่าวมา ยังมีข้อมูลแบบบูลีน เรคอร์ด เซท ไฟล์ พอยน์เตอร์ ฯลฯ ซึ่งข้อมูลแบบสุดท้ายนี้สามารถนำไปประยุกต์เป็นข้อมูลที่ซับซ้อนได้อีก

การแบ่งประเภทข้อมูล สามารถทำได้ดังนี้คือ

ประเภท				
พื้นฐาน	เลขจำนวนจริง	real		สเกลาร์
	เลขจำนวนเต็ม	integer		สเกลาร์
	อักขระ	char		สเกลาร์
	บูลีน	boolean		สเกลาร์
		byte,word		สเกลาร์
		shortint,longint		สเกลาร์
ผู้ใช้สร้างขึ้น	แบบสร้างขึ้นเอง		(บทที่ 5)	สเกลาร์
	แบบกำหนดขอบเขต		(บทที่ 5)	
เป็นโครงสร้าง	อาร์เรย์	array	(บทที่ 4)	
	เรคอร์ด	record	(บทที่ 6)	
	เซท	set	(บทที่ 5)	
	ไฟล์	file	(บทที่ 8)	
	สตริง	string	(บทที่ 4)	
ชี้ข้อมูลอื่น	พอยน์เตอร์	pointer	(บทที่ 10)	

จะเห็นได้ว่า ภาษา Pascal มีข้อมูลตั้งแต่ ประเภทพื้นฐาน เป็นโครงสร้างไปจนถึงพอยน์เตอร์ ซึ่งสามารถแบ่งเป็นประเภทใหญ่ๆได้ 3 ประเภทด้วยกัน

ข้อมูลประเภทพื้นฐานสามารถแบ่งออกได้หลายแบบได้แก่ แบบเลขจำนวนจริง (real)

เลขจำนวนเต็ม (integer) บูลีน (boolean) อักขระ (char) นอกจากนี้ ใน Turbo Pascal ยังมีข้อมูลแบบ byte ซึ่งมีค่าระหว่าง 0..255 ข้อมูลแบบ word ซึ่งมีค่าระหว่าง 0..65535 ข้อมูลแบบ shortint มีค่าระหว่าง -128..127 และข้อมูลแบบ longint ซึ่งมีค่าอยู่ระหว่าง -2147483648..2147483647 ต่อไปเป็นข้อมูลแบบประเภทผู้ใช้สร้างขึ้น ซึ่งแบ่งเป็น 2 แบบได้แก่ แบบสร้างขึ้นเอง (enumerated) และแบบกำหนดขอบเขต (subrange) ข้อมูลทั้งหมดที่กล่าวมานี้ นอกจากเลขจำนวนจริงและแบบกำหนดขอบเขตสามารถเรียกรวมเป็น ข้อมูลแบบสเกลาร์(scalar)

ข้อมูลประเภทเป็นโครงสร้างได้แก่ แบบอาเรย์ (array) เรคอร์ด(record) เซท (set) ไฟล์ (file) และสตริง(string) ซึ่งข้อมูลแบบสตริงไม่มีใน Standard Pascal ถ้าจะใช้ต้องประกาศเป็น array [1..20] of char

ในบทนี้จะกล่าวถึงข้อมูลที่เข้าใจได้ง่ายคือ แบบผู้ใช้สร้างขึ้น แบบบูลีน และแบบเซท

ในบทที่ 1 ได้กล่าวถึงข้อมูลแบบเลขจำนวนเต็ม เลขจำนวนจริง และอักขระแล้ว ส่วนข้อมูลแบบอาร์เรย์และสตริงก็ได้กล่าวในบทที่ 4 สำหรับข้อมูลแบบเรคอร์ด ไฟล์ และ พอยน์เตอร์นั้น เป็นข้อมูลที่ค่อนข้างซับซ้อนซึ่งจะกล่าวในบทต่อไป

< การกำหนดแบบข้อมูลโดยใช้คำสั่ง type >

โดยปรกติข้อมูลที่ไม่ใช่ข้อมูลประเภทพื้นฐาน (integer,real,char,byte เป็นต้น) จะใช้คำสั่ง type เพื่อกำหนดแบบข้อมูล จากบทที่ 4 เมื่อต้องการประกาศให้ a เป็นแบบอาร์เรย์ ก็จะประกาศดังตัวอย่าง

```
var a:array[1..31] of integer;
```

แต่ถ้าใช้คำสั่ง type จะเป็น

```
type day = array[1..31] of integer;
```

เมื่อกำหนดแบบข้อมูลเรียบร้อยแล้ว สามารถประกาศตัวแปร a ได้ดังนี้

```
var a:day;
```

การใช้คำสั่ง type มีประโยชน์ในการกำหนดแบบพารามิเตอร์ซึ่งจะกล่าวในบทที่ 7 เพราะว่าการกำหนดแบบพารามิเตอร์ไม่สามารถกำหนดว่า array[1..31] of integer ลักษณะทั่วไปของคำสั่ง type สามารถกำหนดได้ดังนี้

```
type ชื่อแบบข้อมูล = แบบข้อมูล;
     ชื่อแบบข้อมูล = แบบข้อมูล;
```

จากตัวอย่างข้างบน day ก็คือชื่อแบบข้อมูล array[1..31] of real คือแบบข้อมูล

5.2 ข้อมูลแบบบูลีน

ภาษา BASIC และ C ไม่มีข้อมูลแบบนี้ แต่มีในภาษา Pascal และ FORTRAN ข้อมูลแบบนี้มีค่าเพียง 2 ค่าคือ true กับ false หรือ จริงกับเท็จ การประกาศตัวแปรแบบบูลีนสามารถประกาศได้ดังนี้

var x,y,z:boolean;

ตัวแปรแบบนี้สามารถให้ค่าและใช้โอเปอร์เรเตอร์ได้ดังตัวอย่าง

ex x:=true; (* ให้ x มีค่าเป็นจริง *)

z:=x and y; (* ให้ z มีค่าเป็น a and b *)

ข้อมูลแบบนี้สามารถใช้โอเปอร์เรเตอร์ not,and,or และ xor (xor นั้นไม่มีใน Standard Pascal) ตารางข้างล่างเป็นตารางแสดงผลของการใช้โอเปอร์เรเตอร์ชนิดต่างๆ

x	not x	x	y	x and y	x or y	x xor y
false	true	false	false	false	false	false
true	false	false	true	false	true	true
		true	false	false	true	true
		true	true	true	true	false

ใน Standard Pascal ไม่สามารถส่งเอาท์พุทของข้อมูลแบบนี้โดยใช้คำสั่ง write แต่ใน Turbo Pascal สามารถใช้คำสั่งนี้ได้ จากตัวอย่างข้างบน ถ้าใช้คำสั่ง write(x) ผลก็จะเป็นการแสดง 'FALSE' หรือ 'TRUE' แต่ใน Standard Pascal ต้องใช้วิธีดังที่เห็น

```
if x then write('true')
     else write('false');
```

เนื่องจาก x เป็นตัวแปรแบบบูลีน ดังนั้นจึงไม่จำเป็นต้องเขียนเป็น if x=true ก็ได้ นอกจากนี้ ลำดับของ FALSE กับ TRUE สามารถเปรียบเทียบกันได้อีกด้วย โดยลำดับของ false จะมาก่อน true หรืออาจเขียนเป็นลักษณะ false<true ดังนั้น จึงสามารถเขียนเป็น

```
for x:=false to true do
```

นอกจากนี้ยังสามารถใช้ฟังก์ชัน pred และ succ ซึ่งหมายถึงค่าที่อยู่ข้างหน้าและค่าที่อยู่ถัดไป กล่าวคือ succ(false):=true และ pred(true)=false

ตัวอย่างที่ 5.1 การแสดงผลตารางของ x and y,x or y และ x xor y

```
program bool;
var x,y:boolean;
begin
   writeln('x ':8,'y ':8,'and ':8,'or ':8,'xor ':8);
   writeln('----------------------------------------');
   for x:=false to true do
     for y:=false to true do
       writeln(x:8,y:8,x and y:8,x or y:8,x xor y:8)
end.
```

x	y	and	or	xor
FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	FALSE

1 ประกาศ x,y ให้เป็นข้อมูลแบบบูลีน

2 เปลี่ยนค่า x,y ให้เป็น false กับ true ในระหว่างนั้นก็ให้แสดงค่า x and y, x or y และ x xor y

* สำหรับ Standard Pascal นั้นไม่สามารถแสดงออกมาโดยตรงดังตัวอย่างที่เห็น ถ้าต้องการแสดงเอาพุทของข้อมูลแบบนี้ต้องใช้วิธีดังต่อไปนี้

```
for x:=false to true do
    for y:=false to ture do
        if (x or y) then
            write('TRUE')
        else write('FALSE');
```

5.3 ข้อมูลแบบสร้างขึ้นเอง

ในกรณีที่ข้อมูลที่ต้องการใช้เป็น sun,mon,tue,wed,thu,fri,sat ภาษา Pascal สามารถสร้างข้อมูลขึ้นตามความต้องการของผู้ใช้ วิธีการกำหนดมีดังนี้คือ

```
type day = (sun,mon,tue,wed,thu,fri,sat)
```

เมื่อกำหนดแบบข้อมูลแล้วก็สามารถประกาศตัวแปรได้ดังที่เห็น

```
var x:day;
```

หลังจากนั้น ก็สามารถให้ค่าและเปรียบเทียบลำดับได้เหมือนตัวแปรแบบอื่นๆ ดังนี้คือ

```
x:=sun;
if x=sun then ...
```

ลำดับของข้อมูลจะเป็น

```
sun<mon ... <sat
```

เพราะฉะนั้นจึงสามารถเขียนคำสั่งวนซ้ำเป็นลักษณะที่เห็นข้างล่าง

```
for x:=mon to fri do
```

นอกจากนี้ยังใช้เป็นอินเด็กซ์ในข้อมูลแบบอาร์เรย์ได้อีกด้วย

```
var y : array [sun..sat] of integer;
```

ข้อมูลแบบนี้ไม่สามารถใช้คำสั่ง read หรือ write โดยตรงได้ ถ้าต้องการจะแสดงข้อมูลแบบนี้ก็ต้องใช้คำสั่ง case แทน

ex
```
case x of
    sun:write('sunday');
    mon:write('monday');

    sat:write('saturday')
end;
```

ข้อมูลแบบนี้ เป็นข้อมูลที่ผู้ใช้สร้างขึ้นโดยนำข้อมูลที่ต้องการใช้มาเรียงกัน จึงเรียกข้อมูลอีกชื่อหนึ่งว่า ข้อมูลแบบเรียงลำดับ (enumerated)

ฟังก์ชันที่ใช้กับข้อมูลแบบนี้ได้แก่ pred, succ, ord

ex pred(mon) คือ sun (* ข้อมูลตัวก่อน mon *)

succ(mon) คือ tue (* ข้อมูลตัวหลัง mon *)

ord(mon) คือ 1 (* ลำดับของข้อมูลแรกเป็น 0 *)

```
ex     type day=(sun,mon,tue,wed,thu,fri,sat);
            color=(blue,red,green);
       var x,y:day;
           z:color;
       begin
         for x:=sun to sat do
            if x=sun then ...
            else if z=red ...
       end.
```

ตัวอย่างที่ 5.2 แสดงชื่อและระยะทางของสถานีรถไฟสายใต้

```
program enum;
type station=(huahin,chumpon,suratthani,thung_song,hadyai); .. 1
var a:station;
    distance:array[huahin..hadyai] of integer; ................ 2
    city:array[huahin..hadyai] of string[20];
begin
   city[huahin]         :=    'huahin';
   city[chumpon]        :=   'chumpon';
   city[suratthani]     := 'suratthani';
   city[thung_song]     := 'thung_song';
   city[hadyai]         :=    'hadyai';     ..................... 3
   distance[huahin]     := 229;
   distance[chumpon]    := 485;
   distance[suratthani] := 651;
   distance[thung_song] := 773;
   distance[hadyai]     := 945;
   for a:=huahin to hadyai do
      writeln(city[a]:20,distance[a]:10)  ..................... 4
end.
```

```
     huahin        229
    chumpon        485
 suratthani        651
 thung_song        773
     hadyai        945
```

1 กำหนดให้ stationเป็นข้อมูลของชื่อสถานี ตั้งแต่หัวหินถึงหาดใหญ่

2 a เป็นตัวแปรแบบ station distance เป็นข้อมูลแบบอาร์เรย์แสดงระยะทาง และ city เป็นข้อมูลแบบอาร์เรย์แสดงชื่อสถานี

3 ให้ค่าชื่อและระยะทาง

4 แสดงชื่อและระยะทาง

5.4 ข้อมูลแบบกำหนดขอบเขต

ข้อมูลแบบตัวอักขระ เลขจำนวนเต็ม บูลีน และสร้างขึ้นเอง เรียกรวมกันว่าข้อมูลแบบสเกลาร์ (scalar) ข้อมูลแบบสเกลาร์ คือ ข้อมูลที่มีจำนวนจำกัด และมีลำดับหน้าหลัง ข้อมูลแบบเลขจำนวนจริงมีจำนวนไม่จำกัด จึงไม่ใช่ข้อมูลแบบสเกลาร์

ส่วนข้อมูลแบบกำหนดขอบเขต (subrange) คือ ข้อมูลที่นำเอาส่วนหนึ่งของข้อมูลแบบสเกลาร์มาใช้โดยมีวิธีการกำหนดดังนี้คือ

ตัวแปร : ขอบเขตมากสุด..ขอบเขตน้อยสุด ;

ตัวอย่างเช่น ข้อมูลแบบเลขจำนวนเต็มมีขอบเขตตั้งแต่ -32768 ถึง 32767 ถ้าหากใช้เลขระหว่าง -10..10 หรือ 0..255 ก็สามารถกำหนดแบบข้อมูลขึ้นมาใช้ อินเด็กซ์ของอาร์เรย์ก็คือการใช้ข้อมูลแบบกำหนดขอบเขตได้นั่นเอง

ข้อมูลแบบอักขระก็สามารถกำหนดขอบเขตที่ต้องการใช้ได้เช่นเดียวกัน เช่นถ้าต้องการใช้ตัวอักขระที่เป็นตัวเลขก็กำหนดให้เป็น '0'..'9' หรือตัวอักษรใหญ่ก็กำหนด 'A'..'Z' เป็นต้น

สำหรับข้อมูลแบบสร้างขึ้นเองก็สามารถกำหนดขอบเขตได้เช่นกัน สมมุติว่าข้อมูลเป็น (sun,mon,tue,wed,thu,fri,sat) ถ้าต้องการใช้ข้อมูลเพียงบางส่วนก็สามารถกำหนดได้

การกำหนดแบบข้อมูล โดยปรกติจะใช้คำสั่ง type ดังที่เห็นข้างล่าง

type แบบข้อมูล =(ขอบเขตน้อยสุด..ขอบเขตมากสุด)

```
ex      type day=(sun,mon,tue,wed,thu,fri,sat);
             weekday=mon..fri;  (* weekday มีขอบเขตระหว่าง mon ถึง fri *)
        var x:weekday;
```

เมื่อกำหนดขอบเขตดังตัวอย่างข้างบนแล้ว x จะเป็นตัวแปรที่มีขอบเขตระหว่าง mon ถึง sat และควรระวังอย่าให้ค่า x นอกขอบเขตที่กำหนดไว้เช่น

```
        x:=sun                     (* error *)
```

sun เป็นข้อมูลที่อยู่ภายนอก x ดังนั้นก็จะเกิด error ขึ้น

```
ex      type day=(sun,mon,tue,wed,thu,fri,sat);
             bit=0..1;
             upper='A'..'Z';
             weekday=mon..fri;
             digit=0..9;
```

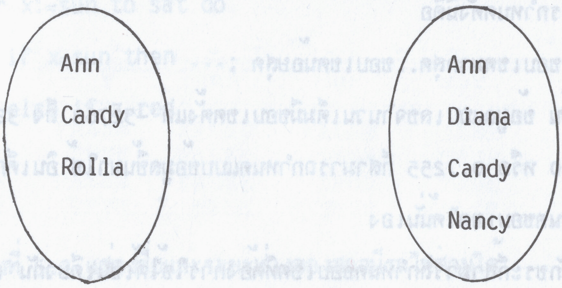
5.5 ข้อมูลแบบเซท

< การกำหนดข้อมูลแบบเซท >

สมมุติว่ามีข้อมูลแบบกำหนดขึ้นเองเป็นดังข้างล่าง

type girl=(Ann,Candy,Diana,Rolla,Nancy)

ถ้าหากว่าต้องการแยกเป็นกลุ่ม a และกลุ่ม b ดังที่เห็น



girla girlb

ในกรณีที่ต้องการให้ girla และ girlb เป็นตัวแปรแบบเซทก็สามารถประกาศได้ดังนี้

```
var gilra,girlb:set of girl;
```

หรืออาจจะกำหนดโดยใช้คำสั่ง type ก็ได้ ซึ่งสามารถเขียนเป็นลักษณะที่เห็นข้างล่าง

```
type girl  = (Ann,Candy,Diana,Nancy,Rolla);
girls = set of girl;
var girla,girlb:girls;
```

เมื่อต้องการจะให้ค่าตัวแปรแบบนี้ สามารถทำได้โดยใส่ค่าเข้าวงเล็บก้ามปู ([]) และใช้คอมมา (,) คั่นระหว่างข้อมูล

```
girla:=[Ann,Cancy,Rolla];
girlb:=[Ann,Candy,Diana,Nancy];
```

ข้อมูลแบบเซทไม่มีการจัดลำดับก่อนหลัง ดังนั้น [Ann,Candy] กับ [Candy,Ann] จึงมีค่าเท่ากัน และยังสามารถใช้แบบกำหนดขอบเขต ซึ่งมีลักษณะเป็น [Ann..Diana]ได้อีกด้วย

การกำหนดแบบข้อมูล สามารถกำหนดได้โดยใช้คำสั่ง set of

type ชื่อแบบข้อมูล = set of แบบข้อมูล;

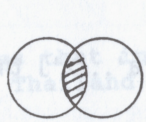
< แบบข้อมูล > ในที่นี้คือ ข้อมูลแบบสเกลาร์และข้อมูลแบบกำหนดขอบเขต จำนวนของสมาชิกของข้อมูลแบบเซทมีได้ไม่เกิน 256

```
ex     type week=set of (sun,mon,tue,wed,thu,fri,sat);
            characters = set of char;
       var  a  : characters;
            b,c : week;
       begin
            a:=['A','S'..'Z'];
            b:=[sun,mon];
            c:=[ ];               <------ ไม่มีสมาชิก
```

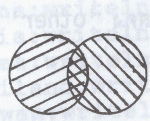
< โอเปอร์เรเตอร์ที่ใช้กับข้อมูลแบบเซท >

โอเปอร์เรเตอร์ที่ใช้กับข้อมูลแบบเซทมีดังนี้คือ

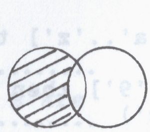
a*b .... สมาชิกที่อยู่ในเซท a และ b (intesection)



a+b .... สมาชิกของเซท a กับ เซท b (union)



a-b .... สมาชิกของเซท a ที่ไม่อยู่ในเซท b (difference)



โอเปอเรเตอร์ * จะดำเนินการก่อน ส่วน + กับ - จะดำเนินการพร้อมกัน

ตัวอย่างเช่น ถ้าเซท a มีสมาชิกเป็น [Ann,Candy,Rolla] และ b มีสมาชิกเป็น [Ann,Diana,Candy,Nancy]

```
a*b ------>    [Ann,Candy]
a+b ------>    [Ann,Candy,Diana,Nancy,Rolla]
a-b ------>    [Rolla]
```

นอกจากนี้ยังสามารถใช้โอเปอร์เรเตอร์ = < > ฯลฯ ได้อีกด้วย

a=b .... เซท a กับ b เท่ากัน (* มีสมาชิกเหมือนกัน *)

a<>b .... เซท a กับ b ไม่เท่ากัน (* สมาชิกของทั้ง 2 เซทไม่เหมือนกัน *)

a<=b .... เซท a อยู่ในเซท b (* สมาชิกของเซท a อยู่ใน เซท b *)

x in a .... x เป็นสมาชิกของ a

โอเปอเรเตอร์ in ช่วยทำให้การตรวจสะดวกขึ้น สมมุติ c เป็นตัวแปรแบบอักขระ ถ้าต้องการตรวจว่า c เป็นตัวอักษรใหญ่หรือไม่โดยปรกติสามารถทำได้ดังนี้

```
if ('A'<=c) and (c<='Z') then ...
```

ถ้าใช้ in ก็สามารถเขียนเป็น

```
if c in ['A'..'Z'] then ...
```

หรือต้องการตรวจว่าเป็นตัวอักษรใหญ่หรืออักษรเล็กก็จะเป็น

```
if c in ['A'..'Z','a'..'z'] then ...
```

หรือต้องการตรวจว่าถ้าไม่ใช่ตัวอักษรก็จะเป็น

```
if not (c in ['A'..'Z','a'..'z'] then ...
```

ตัวอย่างที่ 5.3 ใส่ข้อมูลประเภทอักขระ ถ้าเป็นตัวอักษรให้แสดง 'alphabet' ถ้าเป็นตัวเลขให้แสดง 'numeric' นอกนั้นให้แสดง 'other'

```
program set_type;
var c:char;
begin
   readln(c);
   while c<>'/' do
     begin
       if c in ['A'..'Z','a'..'z'] then
         writeln('alpha')    ................ 1
       else if c in ['0'..'9'] then
         writeln('numeric')  ................ 2
       else
         writeln('other');   ................ 3
       readln(c)
     end
end.
```

1 ดำเนินการเมื่ออินพุทเป็นตัวอักษร

2 ดำเนินการเมื่ออินพุทเป็นตัวเลข

3 ดำเนินการเมื่ออินพุทไม่ได้เป็นทั้งตัวอักษรและตัวเลข

```
5        [enter]
numeric
f        [enter]
alpha
/        [enter]
```

ตัวอย่างที่ 5.4 แสดงประเทศที่มีอยู่ทั้งใน country_a และ country_b และประเทศอยู่ใน country_a หรือ country_b แต่ไม่อยู่ใน country_c

```
program set_type;
type country=(Japan,China,India,Korea,Thailand,Taiwan); .... 1
var a:country;
  country_a,country_b,country_c:set of country; ............ 2
begin
  country_a:=[Japan,Thailand];
  country_b:=[China,India,Thailand];                ............ 3
  country_c:=[China,India];
  writeln('countrys that in set A and B are');
  for a:=Japan to Thailand do
     begin
       if a in (country_a*country_b) then          ............ 4
         case a of
           Japan:writeln('Japan');
           China:writeln('China');
           India:writeln('India');
           Korea:writeln('Korea');
           Thailand:writeln('Thailand');
           Taiwan:writeln('Taiwan')
         end
     end;
  writeln('countrys that in set A or B but C are');
  for a:=Japan to Thailand do
     begin
       if a in (country_a+country_b-country_c) then ....... 5
         case a of
           Japan:writeln('Japan');
           China:writeln('China');
           India:writeln('India');
           Korea:writeln('Korea');
           Thailand:writeln('Thailand');
           Taiwan:writeln('Taiwan')
         end
     end;
end.
```

```
countrys in set A and B are
China
India
Thailand
countrys in set A or B but C are
Japan
Korea
Thailand
```

1 กำหนดให้ country ให้เป็นแบบข้อมูลของประเทศ (Japan,China,India,Korea, Thailand,Taiwan)

2 ประกาศ country_a b และc เป็นตัวแปรแบบ country

3 ให้ค่าสมาชิกกับ country_a,b และ c

4 แสดงประเทศที่อยู่ทั้งในเซท A และ B

5 ประเทศที่อยู่ในเซท a หรือ b แต่ไม่อยู่ใน c

ตัวอย่างที่ 5.5 หาค่าตัวเลขเฉพาะระหว่าง 2 ถึง n

< วิธีคิด >

เนื่องจาก 1 ไม่ถือเป็นตัวเลขเฉพาะ ดังนั้นจึงเริ่มจาก 2

1 กำหนดให้ค่าระหว่าง 2 ถึง n เป็นตัวเลขเฉพาะทุกตัวก่อน

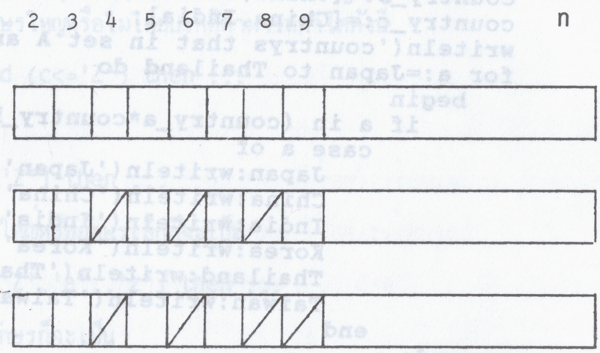
2 เริ่มตรวจจาก 2 ก่อนว่ามีตัวเลขใดที่เป็นจำนวนเท่าของ 2 ถ้ามีก็ตัดทิ้ง

3 ดำเนินการตรวจเลขที่เป็นจำนวนเท่า 3,4,...n โดยใช้วิธีเหมือนข้อ 2

ให้ 2 ถึง n เป็นเลขเฉพาะ

ตัดเลขที่ 2 หารลงตัวออก

ตัดเลขที่ 3 หารลงตัวออก



การเขียนโปรแกรมก็เริ่มจาก การประกาศตัวแปรอาร์เรย์ prime[2] ถึง prime [n] ให้เป็นข้อมูลแบบบูลีน และทำตามขั้นตอนข้างบน โดยให้ตัวแปร prime[] ทุกตัวเป็น true เมื่อถูกตัดออกก็ให้เป็น false

```
program prime1;
const n=1000;
var prime:array[2..n] of boolean;
    i,j:integer;
begin
  for i:=2 to n do
     prime[i]:=true; ................................... 1

  for i:=2 to n do
     if prime[i] then
        for j:=2*i to n do
           if (j mod i)=0 then prime[j]:=false; ... 2

  for i:=2 to n do
     if prime[i] then write(i:4,' ') ................ 3
  end.
```

```
  2   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53
 59  61  67  71  73  79  83  89  97 101 103 107 109 113 127 131
137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223
227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311
313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503
509 521 523 541 547 557 563 569 571 577 587 593 599 601 607 613
617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719
727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827
829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941
947 953 967 971 977 983 991 997
```

1 ให้ 2 ถึง n เป็น true

2 ถ้า prime[i] เป็น true ให้ตรวจว่ามีเลขใดที่หาร prime[ ] ลงตัวหรือไม่ ถ้ามีก็ให้ prime[ ] นั้นเป็น false

3 แสดง prime[i] ที่เป็น true

< วิธีที่ 2 >

กำหนดให้ prime เป็นข้อมูลแบบเซท มีค่าสมาชิก 0 ถึง n และทำวิธีเดียวกับวิธีแรก โดยใช้วิธีการหักออกจากเซท (ใช้โอเปอร์เรเตอร์ -) แต่ข้อมูลแบบเซทมีสมาชิกได้เพียง 256 ตัว จึงหาเลขเฉพาะได้ตั้งแต่ 0 ถึง 255 เท่านั้น

```
program prime2;
const n=255;
var prime :set of 0..n;
    i,j:integer;
begin
   prime:=[2..n];
   for i:=2 to n do
      if i in prime then
         for j:=2*i to n do
            if(j mod i)=0 then prime:=prime-[j];
   for i:=2 to n do
      if i in prime then write(i:3,' ');
   end.
```

ตัวอย่างที่ 5.6 เลือกตัวอักษรอังกฤษ 3 ตัว โดยอย่างน้อยที่สุดต้องมีตัวอักษรตัวใดตัวหนึ่งที่มีอยู่ในเดือนภาษาอังกฤษทั้ง 12 เดือน

< วิธีคิด >

ประกาศ a เป็นอาร์เรย์แบบเซทมีอินเด็กซ์เป็นตัวอักขระ 'a' ถึง 'z' ดังที่เห็นและทำตามขั้นตอนดังนี้ var a:array ['a'..'z'] of set of 1..12;

	1	2	3	4	5	6	7	8	9	
name[1]	j	a	n	u	a	r	y			a['a']=[1]
name[2]	f	e	b	r	u	a	r	y		a['a']=[1,2]
				.						
				.						
name[12]	d	e	c	e	m	b	e	r		

ตรวจตั้งแต่อักษรตัวแรกของเดือนแรก (january) ก็คือ 'j' ให้ a['j'] มีค่าเป็น [1] หมายความว่า เซท a['j'] มีสมาชิกอยู่หนึ่งตัวคือ 1 ต่อไปตรวจตัวถัดไปก็คือ 'a' เซทของ a['a'] ก็จะมีสมาชิกเป็น 1 เช่นกัน และทำเช่นนี้เรื่อยไปจนถึงตัวสุดท้ายของเดือน december หลังจากนั้นก็ตรวจดูว่า 3 เซทใดที่รวมกันได้ [1..12] ขั้นตอนเหล่าถ้าเขียนเป็นลำดับก็สามารถเขียนได้ดังนี้คือ

1 ให้เซท a['a'] ถึง a['z'] มีสมาชิกเท่ากับ 0 กล่าวคือ [ ]

2 ตรวจตั้งแต่อักษรแรกของเดือนแรกว่ามีอักษรใดบ้าง ทำการวนลูปจนครบตั้งแต่อักษรแรกของเดือนแรกจนถึงอักษรสุดท้ายของเดือนสุดท้าย จากรูปข้างบน j คืออักษรแรกของเดือนแรก จึงให้ a['j']:=a['j']+[1] (1 ก็คือเดือนที่ 1) ค่า a['j'] ในตอนต้นกำหนดให้เป็น 0 ([ ]) ดังนั้น a['j'] จะมีสมาชิกคือ 1 ตัวอักษรต่อไปคือ a เพราะฉะนั้น a['a'] จะมีสมาชิกเป็น 1 เช่นกัน เมื่อดำเนินการจนถึงตัวสุดท้ายของเดือนสุดท้าย การให้ค่าจะให้ตั้งแต่ a['a'] ถึง a['z'] การให้ค่า a['a'] ถึง a['z'] ก็เป็นอันสิ้นสุด

3 ตรวจดูว่า มี 3 เซทใดระหว่างเซท a['a'] ถึงเซท a['z'] บ้างที่รวมกันแล้วครบ 12 เดือน กล่าวคือ เซทแรก+เซทที่ 2+เซทที่ 3 = [1..12]

```
program three_char;
type month=1..12;
     letter='a'..'z';
     mset=set of month;
var  a:array[letter] of mset;
     name:array[month] of string[9]; ..................... 1
     l,l1,l2,l3:char;
     m:month;
     i:1..9;
begin
  name[1]  :='january  ';name[2] :='february';
  name[3]  :='march    ';name[4] :='april   ';
  name[5]  :='may      ';name[6] :='june    ';
  name[7]  :='july     ';name[8] :='august  ';
  name[9]  :='september';name[10]:='october ';
  name[11]:='november ';name[12]:='december';

 for l:='a' to 'z' do a[l]:=[ ];      .................... 2
 for m:=1 to 12 do
   for i:=1 to 9 do
      if name[m][i]<>' ' then
      a[name[m][i]]:=a[name[m][i]]+[m]; ................. 3
 for l1:='a' to 'x' do
   for l2:=succ(l1) to 'y' do
     for l3:=succ(l2) to 'z' do
        if a[l1]+a[l2]+a[l3]=[1..12] then .............. 4
          write('(',l1,l2,l3,')',' ');
          writeln
 end.
```

(abj) (abu) (aej) (ael) (aeu) (aey) (ajr) (aru) (mru) (ruy)

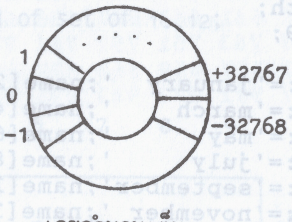
1 ให้ a เป็นอาร์เรย์แบบเซทที่มีอินเด็กซ์เป็นตัวอักขระ 'a' ถึง 'z'

2 ให้ a['a'] ถึง a['z'] ทุกตัวมีสมาชิกเป็น 0 ([ ])

3 ตรวจดูว่าเซท a['a'] ถึงเซท a['z'] มีค่าใดบ้าง

4 แสดง 3 เซทที่บวกกันได้ [1..12]

\# ข้อควรระวังในการใช้ข้อมูลแบบจำนวนเต็มเกินขอบเขต



เลขจำนวนเต็ม

```
program integer_limit;
var i:integer;
begin
   for i:=1 to 5 do
      writeln(i*10000:8)
end.

   10000
   20000
   30000
  -25536
  -15536
```

การให้ค่ากับตัวแปรเกินขอบเขตจะทำให้เกิด error แต่ถ้าให้คำนวณเกินขอบเขตจะไม่เกิด error แต่ทำให้เกิดการคำนวณผิดพลาด ดังรูปจะเห็นได้ว่า ค่าหลังจากเลข 32767 คือ -32768 ในกรณีดังกล่าวควรใช้ข้อมูลแบบ real หรือให้ค่าแก่ตัวแปรโดยตรง จะทำให้คำตอบที่ได้ไม่ผิดพลาด

แบบฝึกหัด

1 จงหาตัวเลขเฉพาะตั้งแต่ 0 ถึง 10000 โดยใช้อาร์เรย์แบบเซท

2 จงหาเลขที่ 2 และ 3 หารลงตัว แต่ 5,7,9 หารไม่ลงตัวโดยใช้ข้อมูลแบบเซท

บทที่ 6

ข้อมูลแบบเรคอร์ด

6.1 ข้อมูลแบบเรคอร์ด

สมมุติว่ามีข้อมูลที่เป็นชื่อ อายุ ที่อยู่ และเบอร์โทรศัพท์ของบุคคล ดังที่เห็นข้างล่าง

	ข้อมูลที่ 1	ข้อมูลที่ 2	ข้อมูลที่ 3
name	Ann	Candy	Mariko
age	20	19	18
country	U.K	U.S.A	Japan
tel	281-6474	235-800	793-9716

จากตัวอย่างข้างบน ถ้าสามารถเก็บข้อมูลเป็นบุคคลๆได้ ก็จะสะดวกในการรวบรวมและค้นหา ซึ่งหลักการนี้เป็นพื้นฐานของข้อมูลแบบเรคอร์ด

ข้อมูลแบบนี้ไม่มีในภาษา BASIC และ FORTRAN แต่มีในภาษา Pascal Cobol และ C ซึ่งเป็นข้อมูลชนิดที่นิยมใช้กันมาก

< การกำหนดแบบข้อมูลเรคอร์ด >

การกำหนดแบบข้อมูลเรคอร์ด สามารถกำหนดโดยใช้คำสั่ง type ได้ดังต่อไปนี้

```
type  person = record                       ข้อมูลแบบ  person
                  name:string[10];          ฟิลด์ name
                  age:integer;              ฟิลด์ age
                  country:string[10];       ฟิลด์ country
                  tel:string[10]            ฟิลด์ tel
               end;
```

เมื่อกำหนดแบบข้อมูลเรียบร้อยแล้ว ก็สามารถกำหนดตัวแปรให้เป็นแบบเรคอร์ดเหมือนกับแบบตัวแปรอื่นๆ เช่น real integer หรือ char เป็นต้น

สมมุติว่าต้องการให้ a เป็นตัวแปรแบบเรคอร์ด (person) ควรกำหนดแบบข้อมูลก่อน หลังจากนั้นก็จึงประกาศตัวแปร a ได้ดังนี้ (หรืออาจประกาศโดยตรงก็ได้)

```
var a:person;
```

การกำหนดแบบข้อมูลเรคอร์ดสามารถเขียนเป็นลักษณะทั่วไปได้ดังนี้ คือ

```
type ชื่อแบบข้อมูล = record
                     ชื่อฟิลด์ : แบบข้อมูล;
                     ชื่อฟิลด์ : แบบข้อมูล;

                   end;
```

ข้อมูลแบบเรคอร์ดมีกี่ฟิลด์ (FILED) ก็ได้ขึ้นอยู่กับผู้ใช้ ในกรณีที่มีฟิลด์เป็นแบบเดียวกันหลายฟิลด์ ก็สามารถใช้คอมมา (,) คั่นได้ดังตัวอย่าง

```
city,country,tel:string[10];
```

สำหรับการกำหนดฟิลด์นั้น ยังสามารถกำหนดแบบข้อมูลเรคอร์ดในฟิลด์ได้อีกด้วย ซึ่งจะมีลักษณะเป็นข้อมูลแบบเรคอร์ดซ้อนในข้อมูลแบบเรคอร์ด

< การอ้างและให้ค่าข้อมูลแบบเรคอร์ด >

จากตัวอย่างที่แล้ว name,age,city,country,tel คือฟิลด์ การอ้างหรือให้ค่าตัวแปรแบบเรคอร์ดสามารถทำได้โดยใช้พีเรียด (.) ดังที่เห็น

ตัวแปรแบบเรคอร์ด.ชื่อฟิลด์

การให้ค่าสามารถทำได้ดังนี้คือ

```
a.name:='Ann';
a.age:=20;
a.country:='U.K';
a.tel:='281-6474'
```

เมื่อให้ค่าตัวแปรแบบเรคอร์ดแล้ว โครงสร้างของตัวแปร a ก็จะมีลักษณะเป็นดังที่เห็น

a.name	Ann
a.age	20
a.country	U.K.
a.tel	281-6474

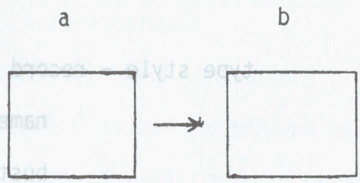
โครงสร้างตัวแปร a ที่เป็นข้อมูลแบบ person

การให้ค่าตัวแปรแบบเรคอร์ด นอกจากจะให้ค่าทีละฟิลด์แล้วยังสามารถให้ทีละเรคอร์ดได้สมมุติว่ากำหนดให้

var a,b:person;

และให้ b:=a;

ก็หมายความว่าค่าของ a ทุกๆฟิลด์จะถูกลอกมาที่ b



ตัวอย่างที่ 6.1 มีตัวแปรแบบเรคอร์ดที่มีฟิลด์เป็น ชื่อ อาชีพ อายุ และให้ค่าแก่ตัวแปรนี้ พร้อมทั้งแสดงเอาท์พุท

```
program record_type;
type person=record
              name:string[20];
              occu:string[20];   ......................... 1
              year:integer;
            end;
var a,b:person;
begin
   a.name:='Candy'; a.occu:='Student'; a.year:=18; ..... 2
   b:=a;                                ................. 3
   write(b.name,'  ',b.occu,'  ',b.year) ................. 4
end.
```

```
  Candy  Student  18
```

1 กำหนดแบบข้อมูลเรคอร์ดมี 3 ฟิลด์คือ name occupation และ age

2 ให้ค่าตัวแปร a ซึ่งสามารถเขียนเป็น

a.name	Candy
a.occupation	student
a.age	19

3 ให้ b มีค่าเหมือน a ทุกฟิลด์

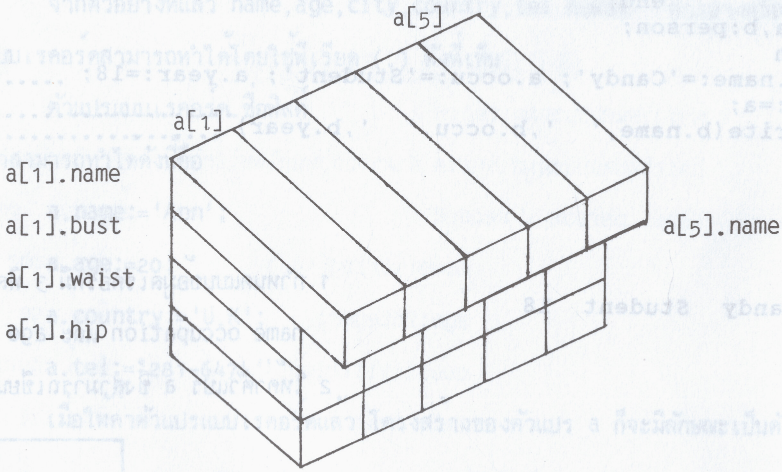
4 แสดงข้อมูลทุกๆฟิลด์

6.2 อาร์เรย์ของข้อมูลแบบเรคอร์ด

อาร์เรย์ของข้อมูลแบบเรคอร์ดสามารถกำหนดและประกาศได้ดังนี้คือ

```
type style = record
                 name:string[20];
                 bust:real;
                 waist:real;
                 hip:real
             end;
var a:array[1..5] of style;
```

โครงสร้างของตัวแปร a จะมีลักษณะดังที่เห็น



การให้ค่าและการอ้างข้อมูลตัวที่ i สามารถเขียนได้ดังที่เห็น

```
a[i].name:='Ann';
a[i].bust:=35.5;
a[i].waist:=24.5;
a[i].hip:=34.0;
```

ตัวอย่างที่ 6.2 ใส่ข้อมูลชื่อ อาชีพ และอายุ ให้กับอาร์เรย์ของตัวแปรแบบเรคอร์ดและแสดงทุกข้อมูล

```
program record_type;
type person=record
               name:string[12];
               occu:string[12];
               year:integer
            end;
var a:array[1..100] of person;   ............................. 1
    i,n:integer;
begin
   n:=1;
   write('name ? ');          readln(a[n].name);
   write('occupation ? '); readln(a[n].occu);
   write('year ? ');          readln(a[n].year);
   writeln;
   while a[n].year <> 0 do          ............................. 2
      begin
         n:=n+1;
         write('name ? ');          readln(a[n].name);
         write('occupation ? '); readln(a[n].occu);
         write('year ? ');          readln(a[n].year);
         writeln
      end;
   for i:=1 to n-1 do
      writeln(a[i].name:12,a[i].occu:12,a[i].year:5) ..... 3
end.
```

```
name ? Candy                [enter]
occupation ? Student        [enter]
year ? 18                   [enter]

name ? Ann                  [enter]
occupation ? Doctor         [enter]
year ? 28                   [enter]

name ? Rolla                [enter]
occupation ? Singer         [enter]
year ? 0                    [enter]

       Candy     Student    18
         Ann      Doctor    28
```

1 ให้ a[1..n] เป็นข้อมูลแบบ person

2 ใส่ข้อมูล (ชื่อ,อาชีพ,อายุ) ให้กับ a[1] จนกว่าอายุเป็น 0

3 แสดงผลออกจอ

6.3 คำสั่ง with

การอ้างหรือให้ค่าฟิลด์ใดฟิลด์หนึ่งของข้อมูลแบบเรคอร์ด โดยปรกติใช้วิธีดังที่เห็น

ตัวแปรแบบเรคอร์ด.ชื่อฟิลด์

แต่ในความเป็นจริง เมื่อเขียนข้อมูลชนิดนี้ถ้าต้องเขียนชื่อตัวแปรทุกครั้งจะเป็นการเสียเวลามาก ดังนั้นในภาษา Pascal จึงมีการใช้คำสั่ง with ซึ่งเมื่อใช้คำสั่งนี้ที่ตัวแปรแบบเรคอร์ดหรือตัวแปรอาร์เรย์แบบเรคอร์ดแล้ว การให้หรืออ้างค่าฟิลด์ภายในบล็อก with สามารถให้หรืออ้างโดยเพียงแต่เขียนชื่อฟิลด์เท่านั้น

คำสั่ง with สามารถเขียนเป็นลักษณะทั่วไปได้ดังนี้คือ

```
with ชื่อตัวแปร,ชื่อตัวแปร do
  begin
                 ...
         การอ้างหรือให้ค่าข้อมูลแบบเรคอร์ดภายในบล็อกนี้ เขียนเพียงชื่อฟิลด์
                 ...
  end;
```

ในกรณีที่มีเพียงข้อความเดียวก็ไม่ต้องใช้คำสั่ง begin..end

การให้ค่าแก่ตัวแปรแบบเรคอร์ด a โดยปรกติจะเขียนในลักษณะ

```
ex   a.name:='Ann';
     a.bust:=35.5;
     a.waist:=24.5;
     a.hip:=34.0;
```

ถ้าใช้คำสั่ง with ก็สามารถเขียนเป็น

```
with a do
  begin
    name:='Ann';
    hip:=35.5;
    waist:=24.5;
    hip:=34.0;
  end;
```

หรือ for i:=1 to m do

with a[i] do

readln(name,bust,waist,hip); ..ให้ค่าแก่ a[i].name..a[i].hip

จะเห็นได้ว่าเมื่อใช้คำสั่ง with แล้ว จะทำให้รวบรัดขึ้น

< ข้อควรระวังการใช้คำสั่ง with >

1) ห้ามให้ค่าแก่อินเด็กซ์ของตัวแปรแบบเรคอร์ดภายในบล็อก with ดังที่เห็น

with a[i] do begin ..; i:=i+1; .. end;

2) คำสั่ง with ที่ใช้ซ้อนกัน

with r_1,r_2,......,rn do s

หมายถึง

with r_1 do with r_2 do ... with rn do s ดังตัวอย่าง

```
ex  type complex= record
                      re,im:real
                  end;
    var z1,z2:complex;
    begin
       with z1,z2 do begin re:=0.0; z1.im:=1.0 end;
```

หมายถึง

```
begin z2.re:=0.0; z1.im:=1.0 end;
```

3) ในกรณีที่ตัวแปรมีชื่อเหมือนกับฟิลด์ การอ้างหรือให้ค่าในบล็อก with หมายถึงฟิลด์

```
ex   var r:record
             a,b:integer
         end;
         a:integer;
     begin
        with r do a:=a+b;
```

หมายถึง

```
r.a:=r.a+r.b;
```

ตัวอย่างที่ 6.3 ใส่ข้อมูลชื่อ หน้าอก เอว และสะโพก ให้ตัวแปรอาร์เรย์แบบเรคอร์ด และแสดงข้อมูลทุกข้อมูล

```
program with_do;
const n=3;
type style=record
          name:string[12];
          bust,waist,hip:real
        end;
var data:array[1..n] of style;
   i:integer;
begin
  for i:=1 to n do
     with data[i] do  ................................. 1
       begin
          write('name ? ');  readln(name);
          write('bust ? ');  readln(bust);
          write('waist ? '); readln(waist);
          write('hip ? ');   readln(hip);
          writeln
       end;
  for i:=1 to n do
     with data[i] do
       writeln(name:12,bust:5:1,waist:5:1,hip:5:1) .. 2
end.
```

```
name ? Candy    [enter]
bust ? 34       [enter]
waist ? 25      [enter]
hip ? 36.5      [enter]

name ? Rolla    [enter]
bust ? 35       [enter]
waist ? 25.5    [enter]
hip ? 35        [enter]

name ? Ann      [enter]
bust ? 36       [enter]
waist ? 24.5    [enter]
hip ? 38        [enter]

       Candy 34.0 25.0 36.5
       Rolla 35.0 25.5 35.0
         Ann 36.0 24.5 38.0
```

1 บล็อกที่อยู่ใน with data[i] do คำสั่ง readln(name) ก็คือ readln(data[i].name)

2 แสดงผล

6.4 ข้อมูลแบบเรคอร์ดที่มีฟิลด์ได้หลายชนิด (variance record)

สมมุติว่ามีข้อมูลแบบเรคอร์ดอยู่ 2 แบบดังที่เห็น

man		woman	
name	ชื่อ	name	ชื่อ
year	อายุ	year	อายุ
height	ส่วนสูง	bust	หน้าอก
weigth	น้ำหนัก	waist	เอว
		hip	สะโพก

เมื่อเปรียบเทียบดูข้อมูลแบบ man กับ woman แล้ว จะเห็นได้ว่า ฟิลด์ที่เหมือนกันคือชื่อกับอายุ ส่วนฟิลด์ที่เหลือจะต่างกัน ในภาษา Pascal สามารถใช้ข้อมูลแบบเรคอร์ดเดียวกันได้ โดยใช้คำสั่ง case เพื่อเลือกแทกฟิลด์ ซึ่งสามารถเขียนได้ดังนี้

```
type  sextype=(man,woman);      (* กำหนดแบบแทกฟิลด์ *)
      person = record
                 name:string[20];
                 year:integer;
                 case sex:sextype of
                      man:(height,weight:real);
                      woman:(bust,waist,hip:real)
               end;
```

ข้อมูลแบบเรคอร์ดข้างบน มีฟิลด์ชื่อ name และ year เป็นฟิลด์ซึ่ง man กับ woman มีเหมือนกัน ส่วนฟิลด์ height,weight ของ man และฟิลด์ bust,waist,hip ของ woman เป็นฟิลด์ที่ไม่เหมือนกัน เมื่อต้องการใช้แบบใดก็สามารถเลือกโดยใช้คำสั่ง case..of แยกเพศหญิงและชาย (man,woman) ส่วน sextype ก็คือเป็นตัวแปรแบบสร้างขึ้นเอง

การกำหนดแบบข้อมูลเรคอร์ดที่มีฟิลด์ได้หลายชนิด สามารถเขียนได้ดังนี้

```
case  แทกฟิลด์ : แบบแทกฟิลด์ (ที่กำหนดตอนต้น) of
        ตัวคงที่ : (ชื่อฟิลด์,ชื่อฟิลด์ ...... :แบบข้อมูล);
        ตัวคงที่ : (ชื่อฟิลด์,ชื่อฟิลด์ ...... :แบบข้อมูล);
```

เมื่อกำหนดแทกฟิลด์เรียบร้อยแล้ว การอ้างหรือการให้ค่าแก่ฟิลด์สามารถทำได้ตามปรกติ ตัวคงที่อาจใช้คอมมา (,) คั่นในกรณีที่ต้องการใช้มากกว่าหนึ่ง ซึ่งจะมีความหมายเหมือนกับโอเปอร์เรเตอร์ or

สมมุติว่า person เป็นข้อมูลแบบเรคอร์ดที่มีฟิลด์ได้หลายชนิดและประกาศให้ a เป็นตัวแปรเรคอร์ดแบบ person ดังข้างล่าง

```
var a:person;
```

การให้ค่าแก่ a สามารถเขียนได้ดังนี้

```
a.sex:=man;              (* เลือกแทกฟิลด์ *)
a.name:='Bill';
a.name:=25;
a.height:=180.5;
a.weight:=72.6;
```

ตัวแปร a จะเป็นตัวแปรแบบ man หรือ woman นั้น ผู้ใช้ต้องเลือกเอง ควรระวังอย่าให้สับสนกัน ด้วยเหตุนี้เองจึงต้องมีแทกฟิลด์เพื่อกำหนดให้เป็นข้อมูลแบบ man หรือ woman โครงสร้างข้อมูลแบบเรคอร์ดชนิดนี้ มีลักษณะดังที่เห็น

แทกฟิลด์

แบบ man	sextype	แบบ woman
lenght	real	bust
weight	real	waist
	real	hip

ข้อมูลแบบ man กับข้อมูลแบบ woman ใช้ตำแหน่งหน่วยความจำเดียวกัน ดังนั้นถ้าให้ a.lenght:=180.5 แต่เมื่ออ้างค่า a.bust ก็จะได้ค่าเดียวกัน ควรระวังด้วยว่าตัวแปรชนิดนี้ไม่สามารถที่จะให้ค่าเป็นแบบ man กับ woman ในเวลาเดียวกันได้ เนื่องจากข้อมูลแบบเรคอร์ดชนิดนี้ สามารถใช้ได้เพียงหนึ่งแบบต่อตัวแปรหนึ่งตัวแปรเท่านั้น

ตัวอย่างที่ 6.4 มีข้อมูลแบบเรคอร์ดที่มีฟิลด์ได้หลายชนิด (แบบ person) จำนวนหนึ่ง ให้แสดงเฉพาะข้อมูลที่มีแทกฟิลด์เป็น woman

```
program union;
const n=3;
type sextype=(man,woman); .................................... 1
    person=record
              name:string[12];
              year:integer;
              case sex:sextype of ........................... 2
                 man:(height,weight:real);
                 woman:(bust,waist,hip:real)
           end;
var data:array[1..n] of person; ............................. 3
   i:integer;
begin
  data[1].sex:=man;
  data[1].name:='Bill';
  data[1].year:=29;
  data[1].height:=180;
  data[1].weight:=75;

  data[2].sex:=woman;
  data[2].name:='Candy';
  data[2].year:=18;                 ......................... 4
  data[2].bust:=34;
  data[2].waist:=25;
  data[2].hip:=32.5;

  data[3].sex:=woman;
  data[3].name:='Rolla';
  data[3].year:=21;
  data[3].bust:=35;
  data[3].waist:=25.5;
  data[3].hip:=33;

  for i:=1 to n do
     with data[i] do
        if sex=woman then ....................................... 5
          writeln(name:10,year:5,bust:6:1,waist:6:1,hip:6:1)
end.
```

```
 Candy     18  34.0  25.0  32.5
 Rolla     21  35.0  25.5  33.0
```

1 กำหนดแบบแทกฟิลด์ชื่อ sextype

2 กำหนดข้อมูลแบบเรคอร์ดที่มีฟิลด์ได้หลายชนิด (แบบ person)

3 กำหนด data ให้เป็นอาร์เรย์ข้อมูลแบบเรคอร์ด

4 ให้ค่าแก่ data[1] ถึง data[3]

5 แสดงข้อมูลที่มีแทกฟิลด์เป็น woman

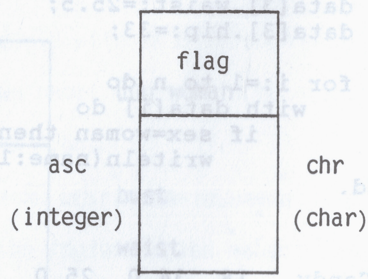
ตัวอย่างที่ 6.5 ใส่ข้อมูลเลขจำนวนเต็ม และแสดงตัวอักขระที่มีรหัส ASCII ตรงกับตัวเลขที่ใส่เข้าไป

```
program union2;
type flag=1..2;                  ................................ 1
     body=record
             case flag of          ............... 2
                1:(asc:integer);
                2:(chr:char)
           end;
var data:body;           ................................ 3
begin
   read(data.asc);       ................................ 4
   writeln(data.asc,' = ',data.chr);
   while data.asc <> 0 do
      begin
         read(data.asc);     ............................ 4
         writeln(data.asc,' = ',data.chr) ............... 5
      end
end.
```

```
56        [enter]
56 = 8
67        [enter]
67 = C
100       [enter]
100 = d
33        [enter]
33 = !
0         [enter]
0 =
```

1 กำหนด flag ให้เป็นชื่อแทกฟิลด์

2 กำหนดให้ body เป็นข้อมูลที่มีโครงสร้างเป็นลักษณะที่เห็น



3 ประกาศ data ให้เป็นข้อมูลแบบ body

4 ใส่ข้อมูล data.asc

5 แสดงอักขระ data.chr ในข้อมูล data ที่ตรงกับข้อมูล data.asc ที่ใส่เข้าไป

ตัวอย่างที่ 6.6 การจัดลำดับข้อมูลแบบเรคอร์ดที่มีฟิลด์ ชื่อ หน้าอก เอว สะโพก โดยจัดลำดับของสะโพกใหญ่ไปหาเล็ก

< วิธีคิด > ใช้การจัดลำดับแบบที่มีอยู่ในตัวอย่างที่ 4.4 และสลับเป็นเรคอร์ดแทนที่จะสลับทีละฟิลด์

```
program record_sort;
const n=5;
type style=record                 ...................... 1
          name:string[12];
          bust,waist,hip:real
        end;
var a:array[1..n] of style;   .................... 2
    dumy:style;
    i,L,k,s:integer;
    max:real;
begin
   writeln('bust waist hip name');
   for i:=1 to n do
      with a[i] do
      readln(bust,waist,hip,name);  ............. 3

   for k:=1 to n-1 do
      begin
         max:=a[k].hip;
         s:=k;
         for L:=k+1 to n do
            if a[L].hip>max then
               begin
                  max:=a[L].hip; s:=L
               end;
         dumy:=a[k]; a[k]:=a[s]; a[s]:=dumy .... 4
      end;
   for i:=1 to n do
      writeln(a[i].name:12,a[i].bust:5:1,
              a[i].waist:5:1,a[i].hip:5:1);
end.
```

```
bust waist hip name
34.0 25.5 36.5 Candy   [enter]
34.5 26.5 35.5 Nancy   [enter]
35.0 25.5 37.0 Rolla   [enter]
35.0 26.5 36.5 Ann     [enter]
34.5 23.5 37.5 Emy     [enter]

     Emy 34.5 23.5 37.5
   Rolla 35.0 25.5 37.0
     Ann 35.0 26.5 36.5
   Candy 34.0 25.5 36.5
   Nancy 34.5 26.5 35.5
```

1 กำหนดข้อมูลแบบ person

2 กำหนดให้ a[1] ถึง a[n]เป็นข้อมูลแบบ person

3 ใส่ข้อมูลให้กับ a[ ] จำนวน n คน

4 สลับคนที่ k กับคนที่ sโดยสลับทั้งเรคอร์ด แทนที่จะสลับทีละฟิลด์

ตัวอย่างที่ 6.7 การคำนวณ + - * / เลขที่เชิงซ้อน

< วิธีคิด > ในภาษา Pascal ไม่มีข้อมูลแบบเลขเชิงซ้อน ดังนั้น จึงต้องสร้างแบบข้อมูลเรคอร์ดที่มีฟิลด์เลขจำนวนจริงและเลขจินตภาพ และทำการคำนวน + - * /

เมื่อกำหนดให้เลขเชิงซ้อน a=x+ui และ b=u+vi การบวกลบคูณหารจะมีผลดังนี้

a+b=(x+u)+(y+v)i

a-b=(x-u)+(y-v)i

a*b=(xu-yv)+(xv+yu)i

a/b=(xu+yv)/(u*u+v*v)+((yu-xv)/(u*u+v*v))i

```
program complex_num;
type complex=record                          ............................ 1
          x:real;
          y:real
       end;
var a,b:complex;                             ............................ 2
    d,e:real;
    c:char;
begin
   write('data ? ');         readln(a.x,a.y);
   write('data ? ');         readln(b.x,b.y);
   write('operator ? ');     readln(c);      .................. 3
   case c of                                           ........... 4
     '+':writeln(a.x+b.x:8:2,a.y+b.y:8:2,'i');
     '-':writeln(a.x-b.x:8:2,a.y-b.y:8:2,'i');
     '*':begin
           d:=a.x*b.x-a.y*b.y;
           e:=a.x*b.y+b.x*a.y;
           writeln(d:8:2,e:8:2,'i')
        end;
     '/':begin
           d:=(a.x*b.x+a.y*b.y)/(b.x*b.x+b.y*b.y);
           e:=(a.y*b.x-a.x*b.y)/(b.x*b.x+b.y*b.y);
           writeln(d:8:2,e:8:2,'i')
        end
   end
end.
```

```
data ? 1 5       [enter]
data ? 2 -6      [enter]
operator ? *     [enter]
   32.00    4.00i
```

1 กำหนดข้อมูลแบบเลขเชิงซ้อน

2 กำหนดตัวแปร a,b ให้เป็นตัวแปรแบบเลขเชิงซ้อน

3 ใส่โอเปอร์เรเตอร์ (+,-,*,/)

4 คำนวนตามโอเปอร์เรเตอร์ที่ใส่

แบบฝึกหัด

1 เขียนโปรแกรมหาคู่ โดยกำหนดแบบข้อมูลเรคอร์ดที่มีฟิลด์ ชื่อนามสกุล วันเดือนปีเกิด เพศและสถานะภาพ (เป็นโสด หรือ แต่งงานแล้ว) และใส่ข้อมูลของบุคคล 2 คน ในกรณีที่บุคคลทั้งสองต่างเพศกัน ชายอายุเกิน 17 ปี หญิงอายุเกิน 15 ปี และสถานะภาพเป็นโสดทั้งคู่ให้เปลี่ยนนามสกุลฝ่ายหญิงและเปลี่ยนสถานะภาพของทั้ง 2 ฝ่าย พร้อมกับแสดงรายละเอียดข้อมูลของทั้งสอง

2 การเรียงลำดับข้อมูลแบบเรคอร์ดดังตัวอย่างที่ 6.6 โดยเรียงลำดับ อก สะโพก เอว จากเล็กไปใหญ่ ในกรณีที่ฟิลด์แรกเท่ากันให้ตรวจฟิลด์ต่อไป (อก สะโพก เอว ตามลำดับ)

3 พนักงานบริษัทหนึ่งต้องการจัดงานสิ้นปีในปลายเดือนธันวาคม โดยจะหักเงิน 5% จากเงินเดือน เมื่อสอบถามพนักงานทุกคนดูแล้วปรากฏว่า พนักงานมีวันว่างไม่ตรงกัน จงเขียนโปรแกรมใส่ข้อมูลวันที่ว่างและเงินเดือนของพนักงาน และทำการคำนวณดูว่า ควรจัดงานวันใดจึงจะรวบรวมเงินค่าจัดงานได้มากที่สุด

บทที่ 7

โพรซิเจอร์และฟังก์ชัน

7.1 โพรซิเจอร์

โพรซิเจอร์คือการดำเนินการข้อความชุดหนึ่ง การใช้โพรซิเจอร์จะช่วยแบ่งโปรแกรมที่ใหญ่ออกเป็นส่วนๆ เพื่อให้การเขียนและตรวจโปรแกรมง่ายและรวบรัดขึ้น นอกจากนี้โพรซิเจอร์ยังเป็นประโยชน์ต่อการดำเนินการข้อความชุดหนึ่งบ่อยๆ ซึ่งเปรียบได้กับ subroutine ของภาษา BASIC ในหนึ่งโปรแกรมอาจมีได้หลายโพรซิเจอร์ ขึ้นอยู่กับขนาดของโปรแกรม

การสร้างและเรียกโพรซิเจอร์นั้นเริ่มจากคำสั่ง procedure ตำแหน่งของโพรซิเจอร์ต้องเขียนก่อนโปรแกรมหลัก เมื่อต้องการใช้โพรซิเจอร์ใดก็เพียงแต่เขียนชื่อโพรซิเจอร์นั้น ดังที่เห็นจากรูปข้างล่าง เมื่อพิจารณาให้ดี โพรซิเจอร์จะมีโครงสร้างเหมือนกับโปรแกรมนั่นเอง สิ่งที่ต่างกันคือวงเล็บหลังชื่อโพรซิเจอร์เท่านั้น

```
      procedure ชื่อโพรซิเจอร์([var]พารามิเตอร์,พารามิเตอร์..); <-+
      ประกาศตัวแปรที่ใช้ในโพรซิเจอร์                              |
         begin                                                |
                                                              |
            ข้อความ:                 ผ่านค่า                     |
                                                         เรียก |
     +---end;                                                 |
     |  begin (* main *)                                      |
กลับ  |                                                        |
     |   ชื่อโพรซิเจอร์(พารามิเตอร์,พารามิเตอร์)  ----------------------+
     +-->
        end.
```

ตัวแปรที่กำหนดในโพรซิเจอร์ใด จะใช้ได้เฉพาะในโพรซิเจอร์นั้น ไม่สามารถนำไปใช้ในโพรซิเจอร์อื่นหรือโปรแกรมหลักได้ ตัวแปรดังกล่าวเรียกว่า ตัวแปรแบบ local (ตรงกันข้าม ตัวแปรที่กำหนดไว้ใช้ในโปรแกรมหลัก ซึ่งสามารถใช้ที่ใดก็ได้เรียกว่า ตัวแปรแบบ global)

< พารามิเตอร์ >

subroutine ในภาษา BASIC จะใช้ตัวแปรของโปรแกรมหลักโดยตรง ซึ่งทำให้การเรียก subroutine แต่ละครั้งไม่ต้องส่งค่าพารามิเตอร์ดังรูป

ex N = 5 GOSUB *SPC (โปรแกรมหลัก)

ใช้ตัวแปร (N) ตัวเดียวกัน

เรียก

กลับ

```
*SPC
FOR K=1 TO N
RETURN
```

(subroutine)

subroutine ในภาษา BASIC เป็นวิธีการที่สะดวกสำหรับผู้เริ่มเขียนโปรแกรมแต่เมื่อพิจารณาให้ดีจะเห็นว่าขาดความอิสระ เพราะต้องใช้ตัวแปรเดียวกับในโปรแกรมหลัก ดังนั้น เมื่อค่าของตัวแปรในโพรซิเจอร์เปลี่ยนไป ค่าของตัวแปรในโปรแกรมหลักก็จะเปลี่ยนไปด้วย ซึ่งมีผลทำให้เกิดการผิดพลาดโดยไม่ได้ตั้งใจ ภาษา Pascal สามารถสร้างโพรซิเจอร์เหมือนกับ subroutine ในภาษา BASIC ได้เช่นกัน แต่โดยปรกติจะส่งข้อมูลผ่านพารามิเตอร์ดังตัวอย่าง

```
ex   procedure spc([var]n:integer); (* n คือ พารามิเตอร์ของโพรซิเจอร์ *)
       begin              แบบพารามิเตอร์
        ข้อความ S              (* ค่า n ในโพรซิเจอร์คือค่า i ในโปรแกรมหลัก *)
       end;
     begin  (* main *)
       i:=5; spc(i);                (* i คือ พารามิเตอร์ของโปรแกรมหลัก *)
     end.
```

จะเห็นได้ว่า เพียงเรียกชื่อโพรซิเจอร์และส่งพารามิเตอร์เข้าในวงเล็บก็เป็นอันสิ้นสุดการผ่านค่า จากตัวอย่างข้างบนโปรแกรมหลักจะส่งพารามิเตอร์ i (มีค่าเป็น 5) ไปที่โพรซิเจอร์และโพรซิเจอร์ก็จะรับค่าทางพารามิเตอร์ n ชื่อพารามิเตอร์ของฝ่ายส่งและรับอาจใช้ชื่อต่างกันได้ ในกรณีที่ต้องการส่งพารามิเตอร์มากกว่าหนึ่ง ก็เพียงแต่ใช้คอมมา (,) คั่นระหว่างพารามิเตอร์

หน้าพารามิเตอร์ของโพรซิเจอร์ จะต้อง var หรือไม่ก็ได้นั้นจะอธิบายภายหลัง

< การกำหนดพารามิเตอร์ >

การกำหนดแบบพารามิเตอร์ สามารถกำหนดได้โดยใส่โคลอน (:) หลังพารามิเตอร์ และใส่แบบข้อมูล แบบข้อมูลอาจเป็น integer,real,char หรือ boolean แต่ไม่สามารถกำหนดข้อมูลที่นอกเหนือไปจากประเภทพื้นฐานดังที่เห็น

```
procedure abc(namename:string[20]);          --> ผิด
procedure def(data:array[1..9] of real);     --> ผิด
```

เนื่องจาก string[20],array [1..9] of real ไม่ได้เป็นข้อมูลประเภทพื้นฐาน ดังนั้นในกรณีที่ต้องใช้พารามิเตอร์นอกเหนือประเภทพื้นฐานเช่น ข้อมูลแบบอาร์เรย์ แบบสร้างขึ้นเอง หรือ แบบเรคอร์ด ต้องกำหนดแบบข้อมูลโดยใช้คำสั่ง type ดังตัวอย่าง

```
ex    type str20=string[20];
           array1=array[1..9] of real;
           day=(sun,non,tue,wed,thu,fri,sat);
           person= record
                        name:string[10];
                        year:integer;
                        city:string[20];
                        tel:string[10]
                   end;
```

เมื่อกำหนดแบบข้อมูลเรียบร้อยแล้ว ก็สามารถกำหนดพารามิเตอร์เป็นแบบข้อมูลที่ต้องการได้ดังตัวอย่าง

```
ex    procedure abc(a:array1:workday:day word:str20);
หรือ  procedure def(var you:person);
```

การกำหนดพารามิเตอร์นั้น สามารถเขียนเป็นลักษณะทั่วไปได้ดังนี้

ชื่อโพรซิเจอร์([var] พารามิเตอร์:แบบข้อมูล.[var] พารามิเตอร์:แบบข้อมูล):

ตัวอย่างที่ 7.1 เขียนโปรแกรมและสร้างโพรซิเจอร์แสดงช่องว่าง

```
program space;
var i:integer;
   procedure spc(n:integer);     .................. 1
     var i:integer;
     begin
       for i:=1 to n do
         write(' ')              .................. 2
     end;

begin
  for i:=1 to 15 do
    begin
      spc(i);                    ............. 3
      writeln('Hello Pascal !')
    end
end.
```

```
 Hello Pascal!
  Hello Pascal!
   Hello Pascal!
    Hello Pascal!
     Hello Pascal!
      Hello Pascal!
       Hello Pascal!
        Hello Pascal!
         Hello Pascal!
          Hello Pascal!
           Hello Pascal!
            Hello Pascal!
             Hello Pascal!
              Hello Pascal!
               Hello Pascal!
```

1 สร้างโพรซิเจอร์โดยให้ n เป็นพารามิเตอร์

i เป็นตัวแปรที่ใช้เฉพาะในโพรซิเจอร์ spc

(ไม่เกี่ยวกับ i ในโปรแกรมหลัก)

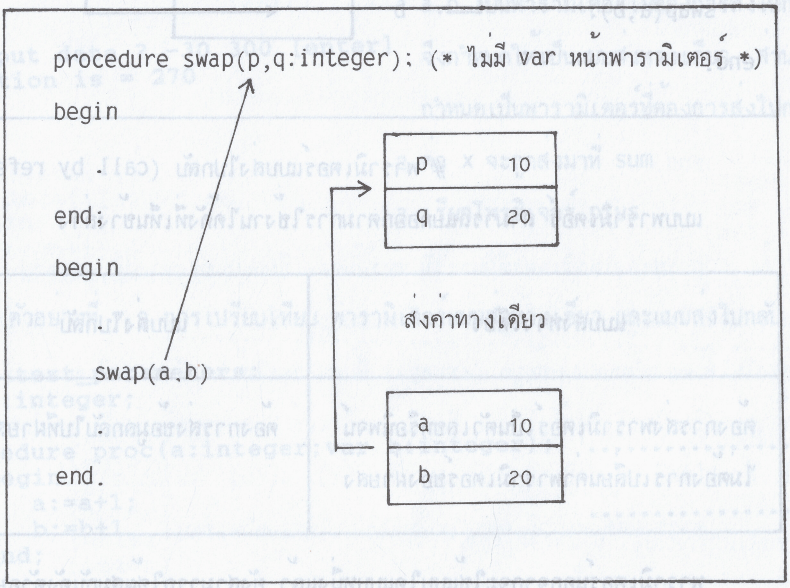
2 เว้นช่องว่าง n ตัว

3 เรียกโพรซิเจอร์ spc

7.2 พารามิเตอร์แบบส่งทางเดียวและแบบส่งไปกลับ

ในภาษา Pascal พารามิเตอร์สามารถแยกเป็นแบบส่งทางเดียว (call by value) และแบบส่งไปกลับ (call by reference) สำหรับความแตกต่างนั้นมีดังนี้คือ

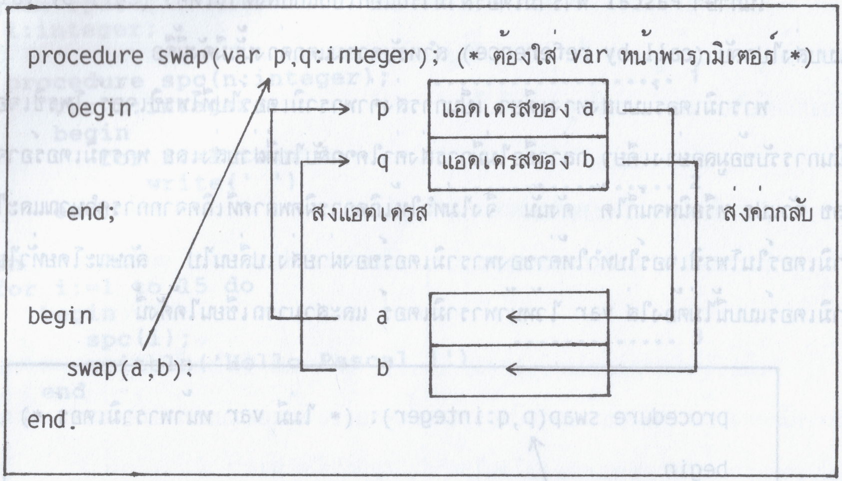
พารามิเตอร์แบบส่งทางเดียว เป็นการส่งค่าพารามิเตอร์ไปที่โพรซิเจอร์ โพรซิเจอร์จะดำเนินการรับข้อมูลอย่างเดียว กล่าวคือไม่มีการส่งค่าใดๆกลับไปที่ฝ่ายส่งเลย พารามิเตอร์อาจเป็นตัวเลข ตัวแปร หรือนิพจนก็ได้ ดังนั้น จึงไม่ทำให้เกิดการผิดพลาดที่เกิดจากการคำนวณและให้ค่าพารามิเตอร์ในโพรซิเจอร์ไปทำให้ค่าของพารามิเตอร์ของฝ่ายส่งเปลี่ยนไป ลักษณะโดยทั่วไปของพารามิเตอร์แบบนี้ไม่ต้องใส่ var ไว้หน้าพารามิเตอร์ และสามารถเขียนได้ดังนี้



\# พารามิเตอร์แบบส่งทางเดียว (call by value)

ในทางตรงกันข้าม การส่งค่าแบบไปกลับนั้น หลังจากที่โพรซิเจอร์ได้รับค่าพารามิเตอร์และดำเนินการคำสั่งในโพรซิเจอร์แล้ว จะส่งค่าพารามิเตอร์กลับไปที่ฝ่ายส่ง เนื่องจากเป็นการส่งแอดเดรส กล่าวคือ แทนที่จะส่งเป็นค่า 1.2 หรือ 10E-01 แตกลับส่งตำแหน่งหน่วยความจำที่เก็บพารามิเตอร์ จึงทำให้ค่าพารามิเตอร์ของฝ่ายส่งเปลี่ยนไป ดังนั้น ในกรณีที่ต้องการให้โพรซิเจอร์ดำเนินการคำนวณและส่งค่าที่คำนวณได้มาที่ฝ่ายส่งต้องใช้พารามิเตอร์แบบนี้ จากรูป พารามิเตอร์แบบส่งไปกลับจะเห็นได้ว่าแอดเดรสของ a กับ b จะถูกส่งมาที่ p กับ q ตามลำดับ หลังจากเรียกโพรซิเจอร์แล้ว ค่า p จะถูกส่งไปที่ a และในทำนองเดียวกันค่า q จะถูกส่งมาที่ b และเนื่องจากเป็นการส่งแบบแอดเดรส ดังนั้น แบบพารามิเตอร์จึงเป็นได้เพียงตัวแปรเท่านั้น

ลักษณะทั่วไปของโพรซิเจอร์แบบนี้ต้องใส่ var ไว้หน้าพารามิเตอร์ดังที่เห็น



พารามิเตอร์แบบส่งไปกลับ (call by reference)

แบบพารามิเตอร์ สามารถแยกออกตามการใช้งานได้ดังที่เห็นข้างล่าง

แบบส่งทางเดียว	แบบส่งไปกลับ
ต้องการส่งพารามิเตอร์เป็นตัวเลขหรือนิพจน์ ไม่ต้องการเปลี่ยนค่าพารามิเตอร์ของฝ่ายส่ง	ต้องการส่งข้อมูลกลับไปที่ฝ่ายส่ง

พารามิเตอร์นอกจากจะใช้แบบใดแบบหนึ่งแล้ว ยังสามารถใช้ผสมกันดังตัวอย่าง

```
procedure plus(a,b:integer,var x:integer);
  begin        ส่งทางเดียว      ส่งแบบไปกลับ
    x:=a+b
  end;
  begin
    plus(10,20,sum); write(sum)     (* sum=30 *)
  end.
```

จากตัวอย่าง 10,20 ต้องกำหนดให้แบบส่งทางเดียว ส่วน sum เป็นแบบพารามิเตอร์ที่ต้องการส่งค่าที่คำนวณได้กลับมาทางฝ่ายส่ง

ตัวอย่างที่ 7.2 ใส่ข้อมูล 2 ข้อมูล และใช้โพรซิเจอร์ทำการหาผลรวมของข้อมูลทั้ง 2

```
program test_plus;
var n,m,sum:integer;

  procedure plus(a,b:integer;var x:integer);     ............... 1
    begin
      x:=a+b                                     ............... 2
    end;

begin
  write('please input data ? ');
  readln(n,m);
  plus(n,m,sum);                                 ............... 3
  writeln('the summation is = ',sum)
end.
```

```
please input data ? -30 300 [enter]
the summation is = 270
```

1 a,b เป็นพารามิเตอร์ไม่ต้องการส่งไปกลับ จึงกำหนดให้เป็นแบบส่งทางเดียว ส่วน x กำหนดเป็นพารามิเตอร์ที่ต้องการส่งไปกลับ

2 ค่า x จะถูกส่งมาที่ sum

3 เรียกโพรซิเจอร์ plus

ตัวอย่างที่ 7.3 การเปรียบเทียบ พารามิเตอร์แบบส่งทางเดียว และแบบส่งไปกลับ

```
program test_parameters;
var x,y:integer;

  procedure proc(a:integer;var b:integer);       ................... 1
    begin
      a:=a+1;                                    ................... 2
      b:=b+1
    end;

begin
  write('input data ');
  readln(x,y);
  proc(x,y);
  writeln('x=',x,'  y=',y)                       ................... 3
end.
```

```
input data 1 1 [enter]
x=1  y=2
```

1 ให้ a เป็นพารามิเตอร์แบบส่งทางเดียว และ b เป็นพารามิเตอร์แบบส่งไปกลับ

2 เพิ่มค่า a และ b จากเดิมอีก 1

3 แสดงค่า x และ y ซึ่งจะเห็นได้ว่า x มีค่าเท่าเดิม ส่วน y มีค่าเพิ่มจากเดิมอีก 1

7.3 การส่งพารามิเตอร์ที่ไม่ใช่ข้อมูลประเภทพื้นฐาน

สมมุติว่ามีตัวแปรอาร์เรย์แบบเลขจำนวนเต็ม a[1] ถึง a[10] และต้องการจะส่งค่า a ทุกค่าไปที่โพรซิเจอร์ abc ก็สามารถเขียนได้ดังนี้

```
type array1=array[1..10] of integer;
var a:array1;
procedure abc([var] b:arrray1);
  begin        ห้ามเขียนเป็น array[1..10] of integer


  end;
begin
  abc(a);
end.
```

การสร้างโพรซิเจอร์ที่มีพารามิเตอร์นั้นต้องเขียนแบบของพารามิเตอร์ด้วย แต่ในกรณีที่พารามิเตอร์ไม่ใช่ข้อมูลประเภทพื้นฐาน จะเขียนแบบข้อมูลเป็น array [1..10] of integer; โดยตรงไม่ได้ ต้องกำหนดแบบข้อมูล ($array_1$) ขึ้นมา และให้ b เป็นพารามิเตอร์แบบ $array_1$ เมื่อต้องการเรียกโพรซิเจอร์ก็เพียงแต่เขียนชื่อโพรซิเจอร์ และข้อมูลที่จะส่งในวงเล็บจากตัวอย่าง b[1]..ถึง b[10] ในโพรซิเจอร์จะมีค่าเท่ากับ a[1]..a[10] ในโปรแกรมหลัก

ตารางแสดงพารามิเตอร์กำหนดแบบและไม่ต้องกำหนดแบบ

ไม่ต้องกำหนดแบบ	ต้องกำหนดแบบ
เลขจำนวนจริง	อาเรย์ สตริง
เลขจำนวนเต็ม	แบบผู้ใช้สร้างขึ้น
บูลีน	เรคอร์ด
อักขระ	เซท
	ไฟล์
	พอยน์เตอร์

ตัวอย่างที่ 7.4 ใส่ข้อมูลอาร์เรย์แบบสตริงซึ่งเป็นชื่อคน จนข้อมูลที่ใส่เป็น '/' จึงถือเป็นการสิ้นสุดการใส่ข้อมูล และสร้างโพรซีเจอร์แสดงข้อมูลทั้งหมด

```
program array_express;
type namedata=array[1..50] of string[20];  ....................... 1
var data:namedata;
    i:integer;

   procedure disp(a:namedata);              ....................... 2
     var j:integer;
     begin
        j:=1;
        while a[j]<>'/' do
           begin
              writeln(a[j]);
              j:=j+1
           end
     end;

begin
   i:=1;
   readln(data[i]);
   while data[i]<>'/' do
      begin
         i:=i+1;
         readln(data[i]);
      end;
   disp(data)
end.          ................................................... 3
```

```
Ann      [enter]
Candy    [enter]
Nancy    [enter]
Bill     [enter]
Jimmy    [enter]
Diana    [enter]
/        [enter]
Ann
Candy
Nancy
Bill
Jimmy
Diana
```

1 กำหนดแบบข้อมูล namedata ให้เป็นแบบข้อมูลอาร์เรย์แบบสตริง

2 สร้างโพรซีเจอร์ที่มี a เป็นพารามิเตอร์แบบ namedata ห้ามกำหนดเป็น (array[1 50] of string[20])

3 เรียกโพรซีเจอร์ disp

ตัวอย่างที่ 7.5 ใส่ข้อมูลอาร์เรย์แบบเรคอร์ดของ 5 คน โดยมีฟิลด์ ชื่อ อายุ ส่วนสูงและน้ำหนักของคน 5 คน และสร้างโพรซิเจอร์แสดงข้อมูลทั้งหมด

```
program record_arg;
const n=5;
type style=record
                name:string[12];
                bust,waist,hip:real
            end;
    stylearray=array[1..n] of style;
var a:stylearray;
    i:integer;

   procedure disp(b:stylearray);
      var j:integer;
      begin
         for j:=1 to n do
            writeln(b[j].name:12,b[j].bust:5:1,
            b[j].waist:5:1,b[j].hip:5:1)
      end;

begin
   writeln('bust waist hip name');
   for i:=1 to n do
      readln(a[i].bust,a[i].waist,a[i].hip,a[i].name);   ........
   disp(a)                                                ........
end.
```

```
 bust waist hip name
 35.5 25.5 36.5 Ann      [enter]
 33.5 24.5 38.0 Rolla    [enter]
 34.5 23.5 37.5 Nancy    [enter]
 35.5 25.5 36.5 Candy    [enter]
 36.0 26.0 38.5 Diana    [enter]

           Ann 35.5 25.5 36.5
         Rolla 33.5 24.5 38.0
         Nancy 34.5 23.5 37.5
         Candy 35.5 25.5 36.5
         Diana 36.0 26.0 38.5
```

1 ใส่ข้อมูลของคน n คนและเก็บในตัวแปร a[ ]

2 เรียกโพรซิเจอร์ disp

7.4 ขอบเขตของตัวแปร

ภายในโปรแกรมหลัก โพรซิเจอร์ หรือ ฟังก์ชัน โดยปรกติแล้วจะต้องมีการใช้ตัวแปรในการดำเนินการ ตัวแปรที่กำหนดขึ้นในบล็อกใด จะมีขอบเขตการใช้งานภายในบล็อกนั้น

สมมุติว่าโปรแกรมมีโครงสร้างดังที่เห็น

```
program test
var j,k,l:integer;          (* ตัวแปรแบบ global *)

   procedure abc(x:real);
      var a,j:integer;   (* ตัวแปรแบบ local *)
      begin

      end;

begin

end.
```

ตัวแปรที่ประกาศตอนต้นโปรแกรม (j.k,l) เป็นตัวแปรที่ใช้ได้ทั้งโปรแกรมหรือเรียกอีกอย่างว่า ตัวแปรแบบ global กล่าวคือ จะใช้ตัวแปรเหลานี้ในโพรซิเจอร์ abc ก็ได้

ส่วนตัวแปรที่ประกาศภายในโพรซิเจอร์ abc กล่าวคือ a,j เป็นตัวแปรที่ใช้ได้เฉพาะในโพรซิเจอร์เท่านั้น ตัวแปรเหล่านี้เรียกอีกอย่างว่า ตัวแปรแบบ local พารามิเตอร์ x ก็ใช้ได้เฉพาะในโพรซิเจอร์เช่นกัน จากตัวอย่าง j เป็นตัวแปรแบบ local จึงใช้ได้เฉพาะในโพรซิเจอร์และไม่มีความสัมพันธ์กับ j ในโปรแกรมหลัก ถ้าหากไม่กำหนดตัวแปร j ในโพรซิเจอร์เมื่อมีการให้ค่า j ก็เป็นการให้ค่า j ในโปรแกรมหลัก

จากเหตุผลดังกล่าว จึงควรหลีกเลี่ยงการผิดพลาดโดยไม่ได้ตั้งใจ กล่าวคือ ภายในโพรซิเจอร์ควรใช้เฉพาะตัวแปรที่กำหนดในโพรซิเจอร์ หรือ ตัวแปรแบบ local เท่านั้น

7.5 ฟังก์ชัน

ฟังก์ชันมีลักษณะคล้ายกับโพรซิเจอร์มาก ข้อแตกต่างระหว่างฟังก์ชันกับโพรซิเจอร์ก็คือ ฟังก์ชันจะมีค่าอยู่ในตัวเอง เช่น sqrt(4) มีค่าเป็น 2 (sin,sqr,sqrt.. เป็นฟังก์ชันมาตรฐาน ซึ่งจะกล่าวในบทที่ 9 สำหรับบทนี้จะกล่าวถึง ฟังก์ชันที่สร้างขึ้นเอง)

< การสร้างและเรียกฟังก์ชัน >

การสร้างฟังก์ชัน เริ่มจากคำสั่ง function เช่นเดียวกับการสร้างโพรซิเจอร์ที่ต้องเริ่มจากคำสั่ง procedure ตามด้วยพารามิเตอร์และแบบฟังก์ชัน เมื่อต้องการเรียกฟังก์ชันก็เพียงเขียนชื่อฟังก์ชันและพารามิเตอร์ก็สามารถเรียกค่าฟังก์ชันได้ดังตัวอย่าง

```
     function ชื่อฟังก์ชัน(พารามิเตอร์,พารามิเตอร์..):แบบฟังก์ชัน; <--+
        กำหนดตัวแปรที่ใช้เฉพาะในฟังก์ชัน                                |
        begin                                                 |
           ข้อความ                                              |
  +---     ชื่อฟังก์ชัน:=....         ผ่านค่า                        เรียก
  |     end;                                                  |
กลับ begin                                                      |
  |                                                           |
  |  ชื่อฟังก์ชัน(พารามิเตอร์,พารามิตอร์) --------------------------+
  +-->   .
     end.
```

ฟังก์ชันเป็นการส่งค่าของฟังก์ชันเอง ดังนั้น ภายในฟังก์ชันจึงต้องมีการให้ค่าแก่ฟังก์ชันนั้นด้วย แนวคิดของฟังก์ชันกับโพรซิเจอร์อาจกล่าวได้ว่าเหมือนกันทุกประการ

สำหรับแบบของฟังก์ชัน ต้องเป็นประเภทพื้นฐานกับแบบพอยน์เตอร์เท่านั้น จะใช้ข้อมูลแบบเป็นโครงสร้างเช่น อาร์เรย์ เรคอร์ด ต่างๆไม่ได้

อนึ่งการผ่านค่าพารามิเตอร์สามารถผ่านในลักษณะของฟังก์ชันได้อีกด้วย

ex

แบบมีพารามิเตอร์

```
function plus(x,y:real):real;
begin
  plus:=x+y
end;
begin
  write(plus(10,20));
end.
```

แบบไม่มีพารามิเตอร์ (x y เป็นตัวแปร global)

```
function plus:real;
begin
  plus:=x+y
end;
begin
  x:=10; y:=20;
  write(plus);
end;
```

ตัวอย่างที่ 7.6 ใส่ข้อมูลตัวเลข ถ้า a เป็นบวกให้แสดง 1 ถ้าเป็น 0 ให้แสดง 0 และถ้าเป็นลบให้แสดง -1 โดยแสดงค่าดังกล่าวโดยใช้ฟังก์ชัน

```
program sign;
var a:integer;
  function sgn(a:integer):integer;  ................ 1
    begin
      if a>0 then
        sgn:=1
      else if a=0 then                ................ 2
        sgn:=0
      else
        sgn:=-1
    end;
begin
  readln(a);
  while a<>-999 do
    begin
      writeln(sgn(a));                ................ 3
      readln(a)
    end
end.
```

```
50      [enter]
1
-100    [enter]
-1
0       [enter]
0
```

1 กำหนดให้ฟังก์ชัน sgn เป็นแบบ integer

2 ค่า sgn จะมีค่าเป็น 1 0 -1 โดยขึ้นกับพารามิเตอร์ที่ส่งเข้ามา

3 เรียกฟังก์ชัน sgn และแสดงค่าฟังก์ชันโดยคำสั่ง writeln

ตัวอย่างที่ 7.7 ใส่ข้อมูลให้กับตัวแปรแบบอาร์เรย์ a[] จนกระทั่งข้อมูลเป็น -999 ต่อจากนั้นใช้ฟังก์ชัน sum(a) เพื่อหาผลรวม

```
program sum_array;
type data=array[1..30] of integer;
var a:data;
   i:integer;

   function sum(b:data):integer; ...................... 1
      var s,i:integer;
      begin
         s:=0;
         i:=1;
         while b[i]<>-999 do
            begin
               s:=s+b[i];                ...................... 2
               i:=i+1
            end;
         sum:=s                          ...................... 3
      end;

begin
   i:=1;
   readln(a[i]);
   while a[i]<>-999 do
      begin
         i:=i+1;
         readln(a[i])
      end;
   writeln('the summation is ',sum(a))   ............ 4
end.
```

1 กำหนดให้ฟังก์ชัน sum เป็นแบบ real

2 ให้ s เป็นผลรวมของ b[ ]

3 ให้ sum เท่ากับ s (ค่าสุดท้าย)

4 เรียกฟังก์ชัน sum

```
1     [enter]
2     [enter]
3     [enter]
4     [enter]
5     [enter]
6     [enter]
7     [enter]
8     [enter]
9     [enter]
10    [enter]
-999  [enter]
the summation is 55
```

7.6 ฟังก์ชันและโพรซิเจอร์ในรูปแบบอื่น ๆ

< โพรซิเจอร์ (ฟังก์ชัน) เรียกโพรซิเจอร์ (ฟังก์ชัน) >

ถ้าหากว่าต้องใช้โพรซิเจอร์ (หรือฟังก์ชัน) a เรียกโพรซิเจอร์ (หรือฟังก์ชัน) b ก็สามารถทำได้ แต่ต้องสร้างโพรซิเจอร์ (หรือฟังก์ชัน) b ก่อนโพรซิเจอร์ (หรือฟังก์ชัน) a

ตัวอย่างที่ 7.8 การคำนวณหาค่า combination ซึ่งคือ วิธีทั้งหมดในการหยิบของ m ชิ้น จากจำนวน n ชิ้น วิธีการหาค่า combination สามารถใช้สูตร

n ชิ้น วิธีการหาค่า combination สามารถใช้สูตร

$$nCm = \frac{n!}{(n-m)!*m!}$$

โดยสร้างฟังก์ชันหาค่าแฟคตอเรียล (fac) ขึ้นมา ต่อจากนั้นสร้างโพรซิเจอร์ combi เพื่อคำนวณค่า combination โดยเรียกฟังก์ชัน fac

```
program combination;
var m,n:integer;
        answer:real;

        function fac(x:integer):real; ..................................................................1
        var L:integer;
            s:real;
        begin
            s: = 1;
            for L: = 1 to x do
                s: = s*L;
            fac: = s
        end;

        procedure combi(a,b:integer;var c:real); ...............................................2
        begin
            c: = fac(a)/fac(b)/fac(a-b)
        end;

begin
        write('enter two numbers ');
        readln(m,n);
        combi(m,n,answer);          ...............................................................3
        writeln('the answer = ',answer:5:0)
end.
```

```
enter two numbers 5 3 |enter|
the answer =    10
```

1 สร้างฟังก์ชัน fac
2 โพรซิเจอร์ combi เรียกฟังก์ชัน fac
3 เรียกโพรซิเจอร์ combi

< โพรซิเจอร์ในโพรซิเจอร์ >

จากตัวอย่างการหาค่า combination ของเลข 2 จำนวน นอกจากจะใช้โพรซิเจอร์เรียกฟังก์ชันแล้ว อาจสร้างฟังก์ชัน (fac) ในโพรซิเจอร์ (combi) ก็ได้ จากตัวอย่างที่แล้วการสร้างฟังก์ชันในโพรซิเจอร์จะเขียนได้ดังตัวอย่างที่ 7.9

ตัวอย่างที่ 7.9 การคำนวณหาค่าแฟคตอเรียลโดยใช้โพรซิเจอร์ในโพรซิเจอร์

```
 program combination;
var m,n:integer;
    a1,a2,a3,answer:real;

  procedure combi(a,b:integer;var c:real);
    var k:real;

    function fac(x:integer):real;
    var L:integer;
        s:real;
        begin
          s:=1;
          for L:=1 to x do
             s:=s*L;
          fac:=s
        end;

  begin
     c:=fac(a)/fac(b)/fac(a-b)
  end;
begin
  write('enter two numbers ');
  readln(m,n);
  combi(m,n,answer);
  writeln('the answer =',answer:10:0)
end.
enter two numbers  5 3 [enter]
the answer =         10
```

ลักษณะของโพรซิเจอร์แบบนี้ มีความแตกต่างจากแบบแรกเล็กน้อย เมื่อเปรียบเทียบดูจะเห็นได้ว่า procedure combi มีลักษณะเหมือนโปรแกรมๆหนึ่งซึ่งมีฟังก์ชัน fac อยู่ภายใน และดูเป็นระเบียบเพราะทำให้ตรวจโปรแกรมได้ง่าย แต่ไม่สามารถเรียกฟังก์ชัน fac นอกโพรซิเจอร์ combi

ตัวอย่างที่ 7.10 การหาตัวคูณรวมน้อยและตัวหารรวมมากของเลข 2 จำนวน

< วิธีคิด >

สมมุติว่าข้อมูลที่ใส่เข้าไปคือ m และ n เมื่อดำเนินการตาม Flowchart จะมีลักษณะดังนี้คือ ถ้าข้อมูลที่ใส่ m มีค่ามากกว่า n แล้วก็ทำการสลับค่า m กับ n หลังจากนั้นให้ w เป็นเศษของ m หาร n ถ้า w ไม่เท่ากับ 0 ให้ m เท่ากับ n และให้ n เท่ากับ w การดำเนินการนี้จะดำเนินจนกระทั่ง w เท่ากับ 0 ซึ่งค่า m สุดท้ายก็คือตัวหารรวมมาก ส่วนตัวคูณรวมน้อยก็คือ ผลคูณของข้อมูลทั้งสองหารด้วยตัวหารรวมมาก ซึ่งเขียนเป็นขั้นตอนได้ดังนี้คือ

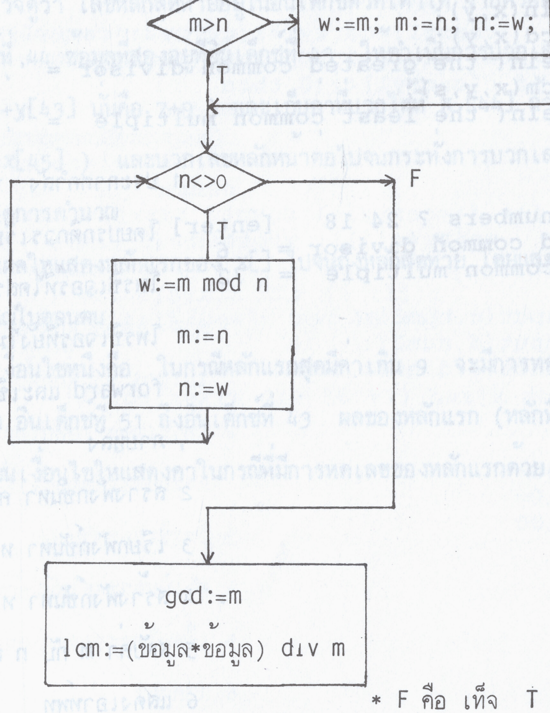
1) ตรวจข้อมูล m กับ n ถ้า m มากกว่า n ให้สลับค่า m กับ n

2) ดำเนินการวนลูป

2.1 ให้ w เท่ากับ m mod n 2.2 ให้ m เท่ากับ n 2.3 ให้ n เท่ากับ w จนกระทั่ง n=0

3) ให้ ห.ร.ม มีค่าเท่ากับ m (ค่าสุดท้าย)

4) ให้ ค.ร.น มีค่าเท่ากับ ผลคูณของข้อมูลหารด้วย ห.ร.ม



* F คือ เท็จ T คือ true

```
program euclid;
var s,t,x,y:integer;

  function gcd(m,n:integer):integer;forward;      .............1
  function lcm(a,b,c:integer):integer;            .............2
    begin
      lcm:=a*b div gcd(a,b)                       .............3
    end;

  function gcd;                                   .............4
  var w:integer;
    begin
      if m<n then
        begin
          w:=m;
          m:=n;
          n:=w                                    .............5
        end;
      while n<>0 do
        begin
          w:=m mod n;
          m:=n;
          n:=w
        end;
      gcd:=m
    end;

begin
  write('enter two numbers ? ');
  readln(x,y);
  s:=gcd(x,y);
  writeln('the greated common divisor = ',s:3);   .........6
  t:=lcm(x,y,s);
  writeln('the least common multiple  = ',t:3);
end.
```

```
enter two numbers ? 24 18    [enter]
the greated common divisor =    6
the least common multiple  =   72
```

1 ประกาศคำสั่ง forward ฟังก์ชัน gcd โดยปรกติการเรียกโพรซิเจอร์นั้น ต้องเรียกโพรซิเจอร์ที่ได้สร้างไว้ล่วงหน้าแล้ว ถ้าเรียกโพรซิเจอร์ที่ยังไม่สร้างต้องประกาศโดยคำสั่ง forward และเขียนโพรซิเจอร์ (gcd) ขึ้นภายหลัง

2 สร้างฟังก์ชันหา ค.ร.น.

3 เรียกฟังก์ชันหา ห.ร.ม.

4 สร้างฟังก์ชันหา ห.ร.ม.

5 สลับค่า m กับ n ถ้า m<n

6 แสดงเอาท์พุท

ตัวอย่างที่ 7.11 การบวกเลข 50 หลักโดยใช้ข้อมูลแบบอาเรย์ที่มีอินเด็กซ์ 1..50

< วิธีคิด >

เก็บข้อมูลเลข 50 หลักในอาร์เรย์ ดังรูปที่เห็น

x []	1 2 3 4 5 6 7
y [] +	9 9 9 9 9 9 9
ผลที่ได้	1 0 1 2 3 4 5 6 6

หลักที่ 50 45 หลักที่ 0

ขั้นตอนการเขียนโปรแกรมสามารถเขียนได้ดังนี้คือ

1 ให้ค่าทุกค่าของ x[] และ y[] เป็น 0 ก่อน

2 อ่านข้อมูลโดยใช้คำสั่ง read(kbd,ข้อมูล) และให้ค่าเลขที่อ่านได้แก่ x[] y[]

3 ทำการตรวจดูว่า เลขหลักสุดท้ายอยู่ในอินเด็กซ์ตัวที่เท่าใด จากตัวอย่าง ข้อมูลแรกมีหลักสุดท้ายที่อินเด็กซ์ที่ 44 ข้อมูลที่สองอยู่ที่อินเด็กซ์ที่ 43 ให้ดำเนินการบวกเลขหลักสุดท้ายของเลขทั้งสอง (x[44]+y[43] นั่นคือ 7+9) และเก็บค่าที่บวกได้ที่ x [44] ถ้าหากเกิน 9 ให้ทดไปที่หลักข้างหน้า (x[45]) และบวกเลขหลักหน้าต่อไปจนกระทั่งการบวกเลขหลักที่ 50 ของทั้งสองตัวแปร จึงสิ้นสุดการคำนวณ

4 การแสดงผลให้แสดงหลักแรกของ x[] ไปจนถึงหลักสุดท้าย โดยแสดงเฉพาะหลักที่มีการให้ค่าจากการคำนวณในตอนต้น

เงื่อนไขอีกเงื่อนไขหนึ่งคือ ในกรณีหลักแรกสุดมีค่าเกิน 9 จะมีการทดไปที่หลักหน้าจากตัวอย่างถ้าแสดงตั้งแต่ อินเด็กซ์ที่ 51 ถึงอินเด็กซ์ที่ 43 ผลของหลักแรก (หลักที่ 52) จะไม่มีการแสดง จึงต้องเขียนเงื่อนไขให้แสดงค่าในกรณีที่มีการทดเลขของหลักแรกด้วย

```
program longdigit;
uses turbo3;                                      ............................ 1
type a=array[0..60] of integer;
var c,d:a;
    i,num1,num2:integer;
  procedure readinput(var x:a;var carry:integer); .......... 2
    var ch:char;
        i:integer;
    begin
      for i:=0 to 60 do x[i]:=0;
      i:=50;
      read(kbd,ch); write(ch);
      repeat
        x[i]:=ord(ch)-ord('0');
        read(kbd,ch); write(ch);
        i:=i-1;
      until not (ch in ['0'..'9']);
      carry:=i+1; writeln
    end;
  procedure calculate(c,d:a;carry1,carry2:integer); ....... 3
    var j,carry:integer;
    begin
      carry:=0;
      repeat
        c[carry1]:=c[carry1]+d[carry2];
        if c[carry1]>9 then
          begin
            c[carry1+1]:=c[carry1+1]+1;
            c[carry1]:=c[carry1]-10               ......... 4
          end;
        carry1:=carry1+1;carry2:=carry2+1;carry:=carry+1;
      until (carry1>50) and (carry2>50);
      if c[carry1]<> 0 then
                                  begin           ......... 5
                                    carry1:=carry1+1;
                                    carry:=carry+1
                                  end;
      for j:=carry1-1 downto carry1-carry do
        write(c[j]); writeln
    end;                                          ......... 6
begin
  readinput(c,num1);
  readinput(d,num2);
  calculate(c,d,num1,num2)
end.
```

```
1234567890+
9876543210=
11111111100
```

1 คำสั่ง read(kbd,ข้อมูล) ต้องใช้คำสั่ง uses turbo3 ควบคู่กันด้วย

2 โพรซีเจอร์ readinput ดำเนินการอ่านตัวอักขระและแปลงเป็นอาร์เรย์เลขจำนวนเต็ม

3 โพรซีเจอร์คำนวณ

4 ดำเนินการทดไปที่หลักหน้า

5 เงื่อนไขตรวจว่า มีการทดไปหลักหน้าหรือไม่

6 แสดงผล

ตัวอย่างที่ 7.12 แสดงตารางเลขฐาน 16 ของเลข 0 ถึง 255

< วิธีคิด >

ความสัมพันธ์ของเลขฐาน 16 กับเลขฐาน 10 มีดังนี้คือ

เลขฐาน 10	0	1	2 ...	9	10	11	12	13	14	15
เลขฐาน 16	0	1	2 ...	9	a	b	c	d	e	f

ถ้าต้องการแปลงเลข 190 ก็สามารถทำได้ดังนี้

190 div 16 = 11 ... b

เศษ = 14 .... e

คำตอบก็คือ be นั่นเอง จากวิธีคำนวณดังกล่าว จะเห็นได้ว่าหลักแรก ก็คือค่าที่ได้จากการหารเลขฐาน 10 ด้วย 16 ส่วนหลักที่สองก็คือเศษที่เหลือ การแปลงจากเลขฐาน 10 เป็นฐาน 16 (เช่น 11 ก็คือ b , 14 ก็คือ e) สามารถใช้ตาราง hex โดยกำหนดให้ hex=string[16] ซึ่งมีเลข 0,1...f อยู่ เมื่อใช้เลขฐาน 10 เป็นอินเด็กซ์ชี้ตัวใดก็จะได้ค่านั้นออกมาเช่น hex[14] ก็คือตัวเลขที่ 14 ของ hex นั่นคือ e

```
program decimaltohex;
const hex:string[16]='0123456789abcdef'; ...................... 1
var j:integer;

   procedure hexdisp(x:integer);
      begin
         if x in [0..255] then
            begin
               if (x mod 6)=5 then
                  begin
                     write(hex[x div 16 +1]);
                     writeln(hex[x mod 16 +1]) ............... 2
                  end
                  else
                     write(hex[x div 16 +1],hex[x mod 16 +1])
                  end
            end;
begin
   for j:=0 to 255 do
      begin
         write(j:4,' = ');
         hexdisp(j)
      end
end.
```

0 = 00	1 = 01	2 = 02	3 = 03	4 = 04	5 = 05
6 = 06	7 = 07	8 = 08	9 = 09	10 = 0a	11 = 0b
12 = 0c	13 = 0d	14 = 0e	15 = 0f	16 = 10	17 = 11
18 = 12	19 = 13	20 = 14	21 = 15	22 = 16	23 = 17
24 = 18	25 = 19	26 = 1a	27 = 1b	28 = 1c	29 = 1d
30 = 1e	31 = 1f	32 = 20	33 = 21	34 = 22	35 = 23
36 = 24	37 = 25	38 = 26	39 = 27	40 = 28	41 = 29
42 = 2a	43 = 2b	44 = 2c	45 = 2d	46 = 2e	47 = 2f
48 = 30	49 = 31	50 = 32	51 = 33	52 = 34	53 = 35
54 = 36	55 = 37	56 = 38	57 = 39	58 = 3a	59 = 3b
60 = 3c	61 = 3d	62 = 3e	63 = 3f	64 = 40	65 = 41
66 = 42	67 = 43	68 = 44	69 = 45	70 = 46	71 = 47
72 = 48	73 = 49	74 = 4a	75 = 4b	76 = 4c	77 = 4d
78 = 4e	79 = 4f	80 = 50	81 = 51	82 = 52	83 = 53
84 = 54	85 = 55	86 = 56	87 = 57	88 = 58	89 = 59
90 = 5a	91 = 5b	92 = 5c	93 = 5d	94 = 5e	95 = 5f
96 = 60	97 = 61	98 = 62	99 = 63	100 = 64	101 = 65
102 = 66	103 = 67	104 = 68	105 = 69	106 = 6a	107 = 6b
108 = 6c	109 = 6d	110 = 6e	111 = 6f	112 = 70	113 = 71
114 = 72	115 = 73	116 = 74	117 = 75	118 = 76	119 = 77
120 = 78	121 = 79	122 = 7a	123 = 7b	124 = 7c	125 = 7d
126 = 7e	127 = 7f	128 = 80	129 = 81	130 = 82	131 = 83
132 = 84	133 = 85	134 = 86	135 = 87	136 = 88	137 = 89
138 = 8a	139 = 8b	140 = 8c	141 = 8d	142 = 8e	143 = 8f
144 = 90	145 = 91	146 = 92	147 = 93	148 = 94	149 = 95
150 = 96	151 = 97	152 = 98	153 = 99	154 = 9a	155 = 9b
156 = 9c	157 = 9d	158 = 9e	159 = 9f	160 = a0	161 = a1
162 = a2	163 = a3	164 = a4	165 = a5	166 = a6	167 = a7
168 = a8	169 = a9	170 = aa	171 = ab	172 = ac	173 = ad
174 = ae	175 = af	176 = b0	177 = b1	178 = b2	179 = b3
180 = b4	181 = b5	182 = b6	183 = b7	184 = b8	185 = b9
186 = ba	187 = bb	188 = bc	189 = bd	190 = be	191 = bf
192 = c0	193 = c1	194 = c2	195 = c3	196 = c4	197 = c5
198 = c6	199 = c7	200 = c8	201 = c9	202 = ca	203 = cb
204 = cc	205 = cd	206 = ce	207 = cf	208 = d0	209 = d1
210 = d2	211 = d3	212 = d4	213 = d5	214 = d6	215 = d7
216 = d8	217 = d9	218 = da	219 = db	220 = dc	221 = dd
222 = de	223 = df	224 = e0	225 = e1	226 = e2	227 = e3
228 = e4	229 = e5	230 = e6	231 = e7	232 = e8	233 = e9
234 = ea	235 = eb	236 = ec	237 = ed	238 = ee	239 = ef
240 = f0	241 = f1	242 = f2	243 = f3	244 = f4	245 = f5
246 = f6	247 = f7	248 = f8	249 = f9	250 = fa	251 = fb
252 = fc	253 = fd	254 = fe	255 = ff		

1 ให้ hex เป็นสตริงมีค่าตั้งแต่ 0 ถึง e

2 hex[x div 16 +1] คือหลักแรกของเลขฐาน 16

hex[x mod 16 +1] คือหลักหลังของเลขฐาน 16

ใน Turbo Pascal สามารถเขียนเลขฐาน 16 ได้โดยตรง โดยใส่เครื่องหมาย $ หน้าเลขนั้น เช่น $10 หมายถึง 16 ในเลขฐาน 10

แบบฝึกหัด

1 จงเขียนโปรแกรมโดยมีฟังก์ชันหา combination ของเลข 2 จำนวน

2 จงเขียนโปรแกรมโดยมีโพรซิเจอร์หาค่า บวก ลบ คูณและหารของเลขเศษส่วน 2 เลข โดยกำหนดให้เลขเศษส่วนเป็นข้อมูลแบบเรคอร์ดที่มีฟิลด์เศษกับฟิลด์ส่วน

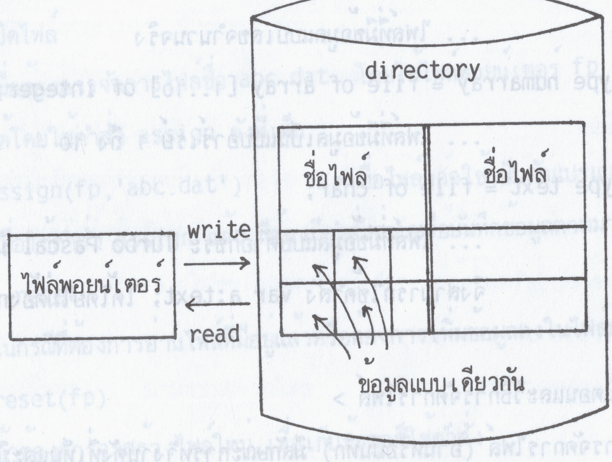
3 จงเขียนโปรแกรมโดยมีสร้างโพรซิเจอร์ บวก ลบ คูณ หาร ของเลข 50 หลัก

บทที่ 8
ข้อมูลแบบไฟล์

8.1 ไฟล์และข้อมูลแบบไฟล์

< ไฟล์และข้อมูลแบบไฟล์ >

ในบทที่ผ่านมาการอ่านและแสดงผลข้อมูล เป็นการอ่านจากคีย์บอร์ดและแสดงผลทางจอ แต่ในกรณีที่ต้องใส่ข้อมูลที่มีจำนวนมาก หรือต้องการบันทึกข้อมูลไว้ใช้งานต่อไปก็ควรจะใช้ข้อมูลแบบไฟล์ ไฟล์ คือส่วนของดิสค์ที่สามารถอ่านหรือบันทึกข้อมูลได้ การดำเนินการของไฟล์มีลักษณะเป็นดังที่เห็นข้างล่าง



ดิสค์

ข้อมูลในไฟล์หนึ่งต้องเป็นข้อมูลแบบเดียวกัน ชื่อไฟล์จะถูกบันทึกอยู่ที่ directory

ภาษา Pascal สิ่งที่ใช้ในการอ่านและบันทึกข้อมูลแบบไฟล์ เรียกว่า ไฟล์พอยน์เตอร์ วิธีการดำเนินการของไฟล์พอยน์เตอร์นั้น สามารถทำความเข้าใจได้ไม่ยาก

< การกำหนดข้อมูลแบบไฟล์ >

การดำเนินการเกี่ยวกับไฟล์นั้นเริ่มจากการกำหนดแบบข้อมูลในไฟล์ที่ต้องการอ่าน หรือบันทึกก่อน การกำหนดข้อมูลแบบไฟล์ สามารถกำหนดโดยใช้คำสั่ง file of ดังที่เห็น

type ชื่อแบบไฟล์ = file of แบบข้อมูล

แบบข้อมูลนั้น สามารถกำหนดให้เป็นข้อมูลแบบใดก็ได้ ยกเว้นแบบไฟล์

ex
```
type seqfile = file of record
                 name:string[20];
                 year:integer
               end;
```

โครงสร้างของไฟล์จะมีลักษณะเป็น

ไฟล์แบบ seqfile	name	year	name	year	....
	ข้อมูลชุดที่ 1		ข้อมูลชุดที่ 2		

ex   type numrec = file of real;

... ไฟล์ที่มีข้อมูลแบบเลขจำนวนจริง

type numarray = file of array [1..10] of integer;

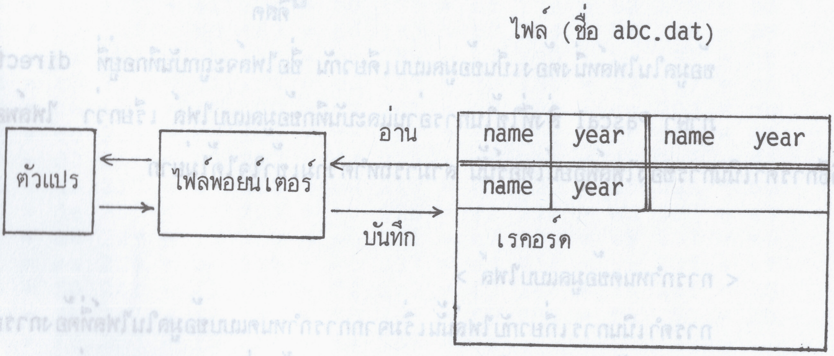
... ไฟล์ที่มีข้อมูลเป็นแบบอาร์เรย์ 1 ถึง 10

type text = file of char;

... ไฟล์ที่มีข้อมูลแบบตัวอักขระ Turbo Pascal มีแบบข้อมูลนี้อยู่ จึงสามารถใช้คำสั่ง var a:text; ได้โดยไม่ต้องกำหนดแบบไฟล์

< ขั้นตอนและวิธีการจัดการไฟล์ >

การจัดการไฟล์ (อ่านหรือบันทึก) มีลักษณะการทำงานดังที่เห็นและมีลำดับดังนี้คือ



1 กำหนดข้อมูลแบบไฟล์

```
type person = record                    (* กำหนดแบบข้อมูล *)
                  name:string[10];
                  year:integer
              end;
     seqfile = file of person;   (* กำหนดแบบไฟล์ *)
```

เมื่อกำหนดแบบข้อมูล (person) แล้ว ก็สามารถกำหนดแบบไฟล์ (seqfile) ได้

2 ประกาศไฟล์พอยน์เตอร์

ในการอ่านหรือบันทึกไฟล์ต้องมีการประกาศไฟล์พอยน์เตอร์ขึ้นมาและใช้ไฟล์พอยน์เตอร์ในการดำเนินการอ่านหรือบันทึกข้อมูล

```
var fp:seqfile;        (* ประกาศไฟล์พอยน์เตอร์ *)
    x:person;          (* ประกาศตัวแปรแบบ person *)
```

3 เปิดไฟล์

เมื่อต้องการจัดการไฟล์ชื่อ abc.dat โดยใช้ไฟล์พอยน์เตอร์ fp เพื่ออ่านหรือบันทึก สามารถเขียนได้โดยใช้คำสั่ง assign ดังที่เห็น

```
assign(fp,'abc.dat')   (* ชื่อไฟล์อาจใช้เป็นตัวแปรแบบสตริงก็ได้ *)
```

เมื่อใช้คำสั่ง assign แล้ว ก็ต้องมีคำสั่งอ่านหรือบันทึกข้อมูลตามมาด้วย โดยใช้คำสั่ง rewrite หรือ reset

ในกรณีที่ต้องการอ่านไฟล์ที่มีอยู่แล้วหรือต้องการเพิ่มข้อมูลลงในไฟล์นั้นก็ใช้คำสั่ง

```
reset(fp)
```

ถ้าต้องการจะสร้างไฟล์ใหม่ เพื่อเก็บข้อมูลก็ใช้คำสั่ง

```
rewrite(fp)
```

4 การอ่านหรือบันทึกข้อมูล

ถ้าให้ x เป็นตัวแปรแบบ person และ fp เป็นไฟล์พอยน์เตอร์แบบ person เมื่อต้องการอ่านหรือบันทึกข้อมูล ก็สามารถทำได้ดังนี้คือ

```
read(fp,x)     (* อ่านข้อมูลจากไฟล์ และให้ค่านั้นกับ x  *)
write(fp,x)    (* บันทึกข้อมูล x ลงไฟล์               *)
```

เมื่ออ่านหรือบันทึกข้อมูลแล้วไฟล์พอยน์เตอร์จะเลื่อนไปที่ตำแหน่งถัดไป การดำเนินการดังกล่าวไฟล์พอยน์เตอร์เป็นตัวควบคุมทั้งหมด

5 การปิดไฟล์

เมื่อทำการอ่านจากไฟล์หรือบันทึกข้อมูลลงไฟล์เรียบร้อยแล้ว ต้องทำการปิดไฟล์โดยใช้คำสั่ง close ดังที่เห็น

```
close(fp)
```

8.2 คำสั่งที่ใช้กับไฟล์

คำสั่งที่ใช้ในการดำเนินการไฟล์มีทั้งโพรซิเจอร์และฟังก์ชั่น ซึ่งมีลักษณะแตกต่างกันตามคอมไพเลอร์ สำหรับคำสั่งที่จะกล่าวเป็นคำสั่งที่ใช้กับ Turbo Pascal

< assign >

การดำเนินการ กำหนดไฟล์พอยน์เตอร์ให้เป็นตัวอ่านหรือบันทึกข้อมูลของไฟล์

แบบ

assign(ไฟล์พอยน์เตอร์,ชื่อไฟล์)

รายละเอียด

เป็นคำสั่งเริ่มต้นการใช้ไฟล์ คำสั่ง assign ดำเนินการกำหนดให้ไฟล์พอยน์เตอร์รับและส่งข้อมูลกับไฟล์ ไฟล์พอยเตอร์ต้องเป็นข้อมูลแบบไฟล์ ชื่อไฟล์อาจใช้ชื่อโดยตรงหรือใช้ตัวแปรแบบสตริง แต่ต้องให้ค่ากับชื่อไฟล์ที่ต้องการดำเนินการก่อนใช้คำสั่ง assign

```
ex  var fp:file of integer;
        filename:string[12];
    begin
       read(filename);
       assign(fp,filename);
```

< reset >

การดำเนินการ เลื่อนตำแหน่งไฟล์พอยน์เตอร์ไปที่ต้นไฟล์ เพื่ออ่านหรือเพิ่มข้อมูล

แบบ

reset(ไฟล์พอยน์เตอร์)

รายละเอียด

เป็นคำสั่งเลื่อนไฟล์พอยน์เตอร์ไปที่ตำแหน่งแรกของไฟล์ที่เปิดโดยคำสั่ง assign ซึ่งเป็นการเตรียมการอ่านหรือบันทึกข้อมูลเพิ่มเติม

ในกรณีที่ไม่มีไฟล์นั้นอยู่ก็จะเกิด I/O error ขึ้นทันที

```
ex  assign(fp,'abc.dat');
    reset(fp);
```

.. เปิดไฟล์เพื่อทำการอ่านหรือบันทึกเพิ่มเติมข้อมูล

< rewrite >

การดำเนินการ　　สร้างไฟล์ใหม่เพื่อทำการบันทึกข้อมูล

แบบ

rewrite(ไฟล์พอยน์เตอร์)

รายละเอียด

เป็นคำสั่งสร้างไฟล์ขึ้นมาใหม่ และเลื่อนไฟล์พอยน์เตอร์ไปที่ตอนต้นสุดของไฟล์ที่เปิดโดยคำสั่ง assign เพื่อเตรียมการบันทึกลงไฟล์นั้น

ในกรณีที่มีไฟล์และข้อมูลอยู่ ข้อมูลภายในไฟล์ก็จะถูกลบทิ้งหมด

```
ex   assign(fp,'abc.dat');
     rewrite(fp);
```

... สร้างไฟล์ abc.dat เพื่อเตรียมการบันทึกข้อมูล

< read >

การดำเนินการ　　อ่านข้อมูลจากไฟล์

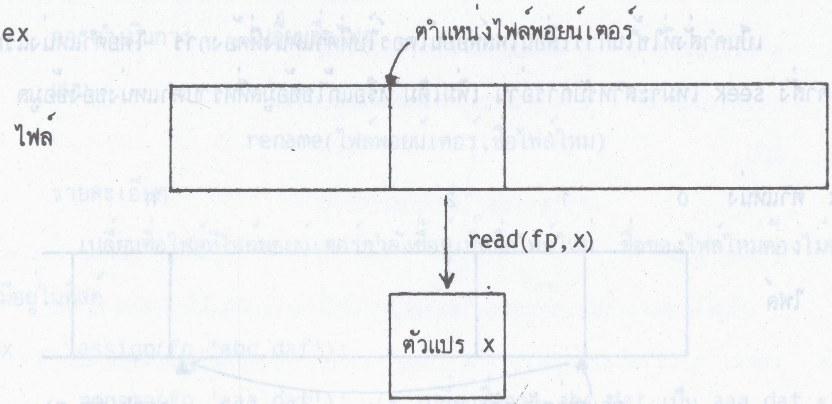
แบบ

read(ไฟล์พอยน์เตอร์,ตัวแปร,ตัวแปร.....)

รายละเอียด

เป็นคำสั่งอ่านข้อมูลในไฟล์ที่ไฟล์พอยน์เตอร์ชี้อยู่ และส่งค่าที่อ่านได้โดยไฟล์พอยน์เตอร์ให้ตัวแปร ตัวแปรดังกล่าวต้องเป็นตัวแปรที่มีแบบข้อมูลเดียวกับไฟล์พอยน์เตอร์ ในกรณีที่ต้องการอ่านข้อมูลให้กับตัวแปรมากกว่าหนึ่งตัวแปร ก็สามารถใช้คอมมา (,) คั่นตัวแปรได้ เมื่ออ่านข้อมูลแล้วไฟล์พอยน์เตอร์จะเลื่อนไปที่ตำแหน่งถัดไป

ex



< write >

การดำเนินการ บันทึกข้อมูลลงในไฟล์

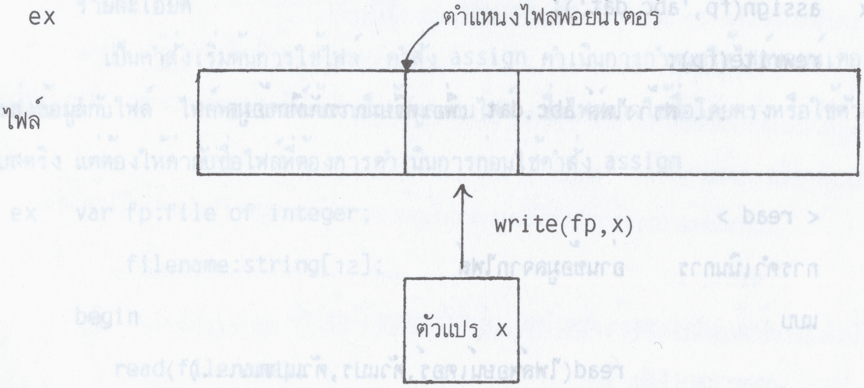
แบบ

write(ไฟล์พอยน์เตอร์,ตัวแปร,ตัวแปร...)

รายละเอียด

เป็นคำสั่งบันทึกข้อมูลลงในไฟล์ที่ตำแหน่งไฟล์พอยน์เตอร์ชี้อยู่ ตัวแปรจะต้องเป็นข้อมูลเช่นเดียวกันกับไฟล์พอยน์เตอร์ ในกรณีที่ตัวแปรหลายตัวก็สามารถใช้คอมมา(,)คั่น เมื่อบันทึกข้อมูลแล้ว ไฟล์พอยน์เตอร์ก็จะเลื่อนไปที่ตำแหน่งถัดไปเพื่อเตรียมการบันทึกข้อมูลต่อไป

ex



< seek >

การดำเนินการ เลื่อนไฟล์พอยน์เตอร์ไปที่ตำแหน่งที่ต้องการ

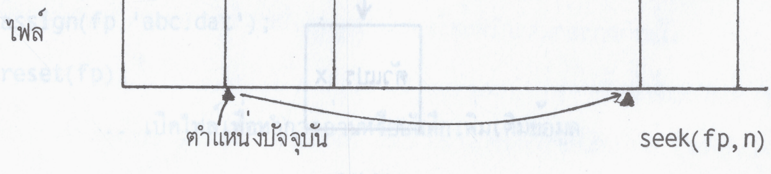
แบบ

seek(ไฟล์พอยน์เตอร์,ตำแหน่งที่ต้องการ)

รายละเอียด

เป็นคำสั่งที่ใช้ในการเลื่อนไฟล์พอยน์เตอร์ไปที่ตำแหน่งที่ต้องการ โดยตำแหน่งแรกสุดคือ 0 คำสั่ง seek เหมาะสำหรับการอ่าน เพิ่มเติม หรือแก้ไขข้อมูลที่ทราบตำแหน่งของข้อมูล

ex



< close >

การดำเนินการ ปิดไฟล์

แบบ

close(ไฟล์พอยน์เตอร์)

รายละเอียด

เป็นคำสั่งสิ้นสุดการอ่านหรือบันทึกไฟล์ ในกรณีที่มีการใช้ไฟล์ใหม่ ชื่อไฟล์จะถูกบันทึกใน directory บนดิสค์

ในการเปิดไฟล์เพื่ออ่านหรือบันทึกข้อมูล ต้องปิดไฟล์ด้วยคำสั่ง close ทุกครั้ง

< erase >

การดำเนินการ ลบไฟล์ที่ไม่ต้องการทิ้ง

แบบ

erase(ไฟล์พอยน์เตอร์)

รายละเอียด

ลบไฟล์ที่ไฟล์พอยน์เตอร์ชี้อยู่จาก directory ถ้าหากใช้คำสั่ง reset หรือ rewrite ควรปิด (close) ไฟล์นั้นก่อนใช้คำสั่ง erase

ex assign(fp,'abc.dat');

erase(fp);

... ลบไฟล์ abc.dat ออกจาก directory

< rename >

การดำเนินการ เปลี่ยนชื่อไฟล์

แบบ

rename(ไฟล์พอยน์เตอร์,ชื่อไฟล์ใหม่)

รายละเอียด

เปลี่ยนชื่อไฟล์ที่ไฟล์พอยน์เตอร์กำลังชี้อยู่เป็นชื่อไฟล์ใหม่ ชื่อของไฟล์ใหม่ต้องไม่ซ้ำกับไฟล์ที่มีอยู่ในดิสค์

ex assign(fp,'abc.dat');

rename(fp,'aaa.dat'); (* เปลี่ยนชื่อจาก abc.dat เป็น aaa.dat *)

< eof > (ฟังก์ชันแบบบูลีน)

การดำเนินการ ตรวจดูตำแหน่งไฟล์พอยน์เตอร์ว่าอยู่ท้ายสุดของไฟล์หรือไม่

แบบ

eof(ไฟล์พอยน์เตอร์)

รายละเอียด

ตรวจดูว่าไฟล์พอยน์เตอร์ชี้ไปที่ตำแหน่งสุดท้ายของไฟล์หรือไม่ ถ้าไฟล์พอยน์เตอร์ชี้ไปที่ตำแหน่งสุดท้าย eof() จะเป็น true ถ้าไม่ใช่ก็จะเป็น false

```
ex    while not eof(fp) do
        begin

        end;       (* ดำเนินการจนกว่าไฟล์พอยน์เตอร์จะชี้ไปที่ตำแหน่งสุดท้าย *)
```

< filepos > (ฟังก์ชันแบบเลขจำนวนเต็ม)

การดำเนินการ ส่งตำแหน่งไฟล์พอยน์เตอร์

แบบ

filepos(ไฟล์พอยน์เตอร์)

รายละเอียด

เป็นฟังก์ชันส่งตำแหน่งที่ไฟล์พอยน์เตอร์ชี้อยู่กลับมา ตำแหน่งแรกมีค่าเป็น 0 และค่าที่ฟังก์ชันส่งกลับมาเป็นเลขจำนวนเต็ม

```
ex    write(filepos(fp))  (* แสดงตำแหน่งที่ไฟล์พอยน์เตอร์ชี้อยู่ *)
```

< filesize > (ฟังก์ชันแบบเลขจำนวนเต็ม)

การดำเนินการ ส่งจำนวนข้อมูลทั้งหมดที่มีอยู่ในไฟล์

แบบ

filesize(ไฟล์พอยน์เตอร์)

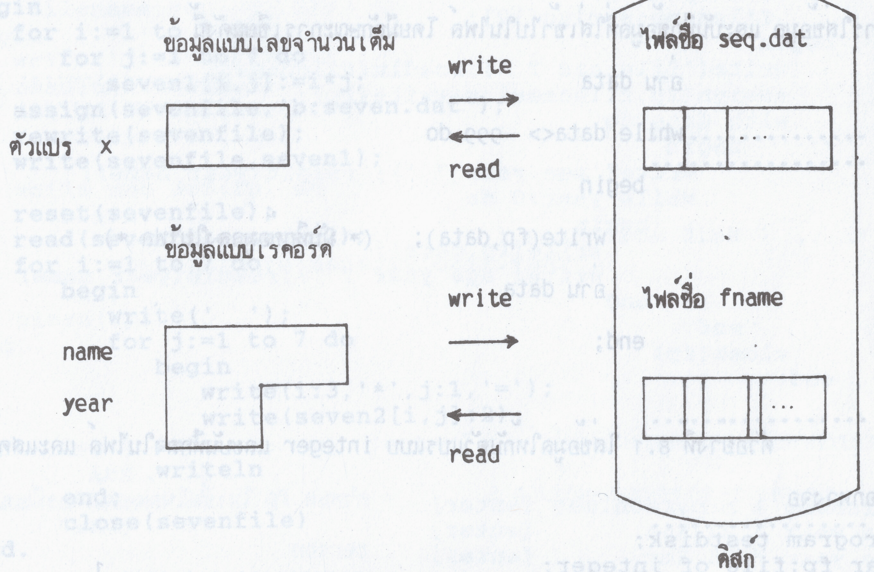
รายละเอียด

เป็นฟังก์ชันส่งจำนวนข้อมูลของไฟล์กลับมา ค่าที่ส่งกลับมาเป็นเลขจำนวนเต็ม

```
ex    seek(fp,filesize(fp))  (* เลื่อนไฟล์พอยน์เตอร์ไปที่ตำแหน่งสุดท้าย *)
```

8.3 ซีเควนเชียลไฟล์ (sequencial file)

สมมุติว่ามีข้อมูลอยู่ชุดหนึ่ง และต้องการอ่านหรือบันทึกข้อมูลดังกล่าวลงในไฟล์ โดยลำดับข้อมูลในไฟล์ดังกล่าวเริ่มตั้งแต่ข้อมูลแรกสุดไปจนถึงข้อมูลสุดท้าย วิธีการอ่านและบันทึกไฟล์ดังกล่าวคือการจัดการไฟล์แบบ "ซีเควนเชียลไฟล์"(sequencial file) ซึ่งมีลักษณะดังนี้



ถ้าเป็นข้อมูลประเภทพื้นฐานก็สามารถกำหนดไฟล์พอยน์เตอร์ดังที่เห็น

```
var f:file of integer;
```

แต่ถ้าเป็นข้อมูลประเภทอื่นควรกำหนดแบบก่อน เช่น กำหนดข้อมูลแบบ person หลังจากนั้นก็กำหนด fp ให้เป็นไฟล์พอยน์เตอร์แบบ person ดังตัวอย่าง

```
ex)    type person=record
                       name :string[20];
                       year :integer
                     end;
       var fp:file of person;
```

หรือจะกำหนดแบบไฟล์โดยคำสั่ง type เป็น

```
seqfile:file of person;
```

หลังจากนั้นจึงกำหนดไฟล์พอยน์เตอร์ก็ได้

```
var fp:seqfile;
```

< การบันทึกข้อมูลลงในซีเควนเชียลไฟล์ >

สมมติว่าไฟล์ชื่อ fname ถ้าต้องการเปิดไฟล์เพื่อบันทึกข้อมูลก็สามารถเขียนได้ดังนี้

```
assign(fp,fname);
   rewrite(fp);
```

การใส่ข้อมูลก็สามารถใส่ข้อมูลโดยกำหนดเงื่อนไขว่า data มีค่าเป็น -999 ให้สิ้นสุดการใส่ข้อมูล และบันทึกข้อมูลที่ใส่เข้าไปในไฟล์ โดยมีลักษณะการเขียนดังนี้

```
อ่าน data
while data<> -999 do
   begin
      write(fp,data);   (* บันทึกข้อมูลลงในไฟล์ *)
      อ่าน data
   end;
```

ตัวอย่างที่ 8.1 ใส่ข้อมูลให้กับตัวแปรแบบ integer และบันทึกลงในไฟล์ และแสดงค่าออกทางจอ

```
program testdisk;
var fp:file of integer;                ................... 1
    filename:string[15];
    i:integer;
begin
   write('filename ? ');
   read(filename);
   assign(fp,filename);
   rewrite(fp);                        ................... 2
   readln(i);
   while i<>-999 do                    ................... 3
      begin
         write(fp,i);
         readln(i)
      end;
   close(fp)
end.
   filename ? b:test.num [enter]
   56     [enter]
   78     [enter]
   55     [enter]
   100    [enter]
   -999   [enter]
```

1 กำหนดให้ fp เป็นไฟล์พอยน์เตอร์

2 เปิดไฟล์เพื่อบันทึกข้อมูล

3 ใส่และบันทึกข้อมูลจนกว่า ข้อมูลเป็น -999

ข้อมูลในไฟล์จะเป็น 56 78 55 100

ตัวอย่างที่ 8.2 เขียนโปรแกรมสร้างสูตรคูณ ตั้งแต่ 1*1 ถึง 7*7 บันทึกลงดิสค์ และแสดงออกจอ

```
program multiseven;
type seven=array[1..7,1..7] of integer;          ................. 1
var seven1,seven2:seven;                         ................. 2
  sevenfile:file of seven;                       ................. 3
  i,j:integer;
begin
  for i:=1 to 7 do
    for j:=1 to 7 do
      seven1[i,j]:=i*j;
  assign(sevenfile,'b:seven.dat');
  rewrite(sevenfile);                            ................. 4
  write(sevenfile,seven1);                       ................. 5
  reset(sevenfile);
  read(sevenfile,seven2);
  for i:=1 to 7 do
    begin
      write('  ');
       for j:=1 to 7 do
         begin
           write(i:3,'*',j:1,'=');
           write(seven2[i,j]:2)                  ................. 6
         end;
         writeln
    end;
    close(sevenfile)                             ................. 7
end.
```

```
1*1= 1  1*2= 2  1*3= 3  1*4= 4  1*5= 5  1*6= 6  1*7= 7
2*1= 2  2*2= 4  2*3= 6  2*4= 8  2*5=10  2*6=12  2*7=14
3*1= 3  3*2= 6  3*3= 9  3*4=12  3*5=15  3*6=18  3*7=21
4*1= 4  4*2= 8  4*3=12  4*4=16  4*5=20  4*6=24  4*7=28
5*1= 5  5*2=10  5*3=15  5*4=20  5*5=25  5*6=30  5*7=35
6*1= 6  6*2=12  6*3=18  6*4=24  6*5=30  6*6=36  6*7=42
7*1= 7  7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49
```

1 กำหนดให้ seven เป็นอาร์เรย์ตั้งแต่ [1,1] ถึง [7,7] เป็นแบบ integer

2 ให้ seven1 เป็นตัวแปรแบบ seven

3 ให้ sevenfile เป็นไฟล์พอยน์เตอร์แบบ seven

4 เปิดไฟล์เพื่อทำการบันทึกข้อมูลแบบอาร์เรย์

5 write(sevenfile,seven1) หมายถึงบันทึกข้อมูลจาก seven1[1,1] ถึง seven1[7,7])

6 แสดงออกจอ

7 ปิดไฟล์

ตัวอย่างที่ 8.3 บันทึกข้อมูลแบบ person ลงดิสก์จนกว่า ฟิลด์อายุเป็นลบ

```
program writefile;
type person=record
            name:string[10];
            year:integer
        end;
var fp:file of person;  ......................... 1
    data:person;
    filename:string[12];
begin
   write('filename ? ');readln(filename); .............. 2
   assign(fp,filename);rewrite(fp); ................ 3
   with data do
      begin
         write('age year ? '); readln(year,name);
         while year>0 do
            begin
               write(fp,data);
               write('age year ? ');readln(year,name)
            end
      end;
   close(fp)
end.
```

```
filename ? b:person.seq [enter]
age year ? 22 Ann       [enter]
age year ? 25 Emy       [enter]
age year ? 26 Candy     [enter]
age year ? -1 /         [enter]
```

1 กำหนด fp ให้เป็นไฟล์พอยน์เตอร์มีแบบเป็น person

2 อ่านชื่อไฟล์ที่ต้องการบันทึกข้อมูล

3 เปิดไฟล์และเริ่มทำการบันทึกข้อมูลโดยบันทึกได้ทีละเรคอร์ด จนกระทั่งข้อมูลมีอายุเป็นลบ

< การอ่านซีเควนเชียลไฟล์ >

คำสั่งที่ใช้ในการเปิดและอ่านไฟล์ชื่อ fname สามารถเขียนได้ดังนี้คือ

```
assign(fp,fname);
reset(fp);
```

ถ้าต้องการให้อ่านข้อมูลทุกข้อมูลจนถึงข้อมูลสุดท้าย สามารถเขียนได้เป็นลักษณะ

```
while not eof(fp) do
   begin
      read(fp,dat);  (* อ่านข้อมูลจากไฟล์ *)
   end;
```

ตัวอย่างที่ 8.4 อ่านข้อมูลจากไฟล์ที่สร้างขึ้นจากตัวอย่างที่ 8.3 และแสดงออกจอ

```
program readfile;
type person=record
              name:string[10];
              year:integer
            end;
var fp:file of person;
    data:person;
    filename:string[12];
begin
   write('filename ? ');
   readln(filename);
   assign(fp,filename);
   reset(fp);                          { open to read } ........ 1
   while not eof(fp) do
      begin
        read(fp,data);                 { read from disk } ..... 2
        writeln(data.name:10,data.year:5)
      end;
   close(fp)
end.
```

```
filename ? b:person.seq   [enter]
      Ann     22
      Emy     25
    Candy     26
```

1 เปิดไฟล์ person.seq เพื่อทำการอ่าน

2 อ่านข้อมูลและแสดงทุกข้อมูลทางจอ

< การต่อเติมข้อมูลลงในซีเควนเชียลไฟล์ >

คำสั่ง rewrite เป็นคำสั่งสำหรับการสร้างไฟล์ขึ้นมาใหม่ ในกรณีที่ต้องการเพิ่มเติมข้อมูล โดยให้ข้อมูลเก่าเหลืออยู่ การต่อเติมข้อมูลแบบนี้สามารถทำได้โดยใช้คำสั่ง reset,seek ดังที่เห็นข้างล่าง

```
assign(fp,fname);
reset(fp);
seek(fp,filesize(fp));
```

....เลื่อนตำแหน่งไฟล์พอยน์เตอร์ไปที่ท้ายสุดของไฟล์

ตัวอย่างที่ 8.5 ต่อเติมข้อมูลในไฟล์ที่สร้างขึ้นจากตัวอย่างที่ 8.3

```
program appendfile;
type person=record
              name:string[10];
              year:integer
          end;
var fp:file of person;
    data:person;
    filename:string[12];
begin
   write('filename ? ');
   readln(filename);
   assign(fp,filename);
   reset(fp);                                        ..............1
   seek(fp,filesize(fp));                            ..............2

   with data do
     begin
        write('age year ? ');
        readln(year,name);
        while year>0 do                              ..............3
          begin
             write(fp,data);                   { write to disk }
             write('age year ? ');
             readln(year,name)
          end
     end;
   close(fp)
end.
```

```
filename ? b:person.seq   [enter]
age year ? 28 Nancy       [enter]
age year ? 31 Eluza       [enter]
age Year ? -1 Nobody      [enter]
```

1 เปิดไฟล์ชื่อ person.seq เพื่อเพิ่มเติมข้อมูล

2 เลื่อนตำแหน่งไฟล์พอยน์เตอร์ไปที่ท้ายไฟล์

3 เพื่อเติมข้อมูลและบันทึกลงในไฟล์จนกว่าอายุจะน้อยกว่า 0

ในไฟล์ person.dat จะมีข้อมูลดังข้างล่าง (ลองใช้ตัวอย่างที่ 8.4 เพื่ออ่านดู)

```
Ann     22
Emy     25
Candy   26
Nancy   28
Eluza   31
```

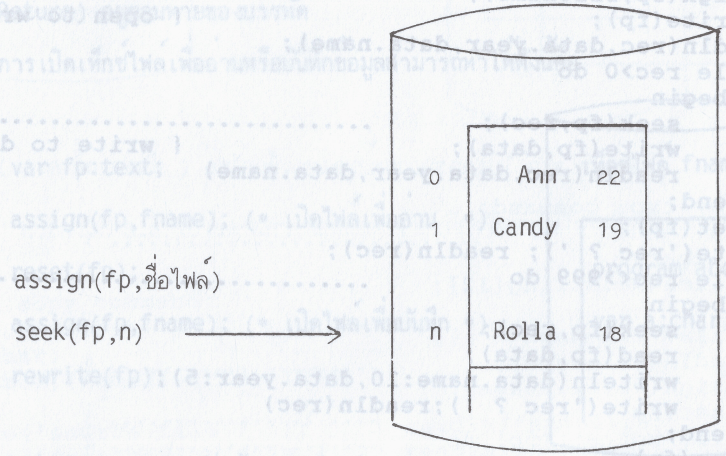
8.4 แรนดัมไฟล์ (random file)

ซีเควนเชียลไฟล์เป็นการดำเนินการอ่านหรือบันทึกข้อมูลแบบไฟล์ โดยเริ่มตั้งแต่ข้อมูลแรกไปจนถึงข้อมูลสุดท้าย ไม่สามารถอ่านหรือบันทึกข้อมูลตำแหน่งที่ต้องการได้

แต่สำหรับแรนดัมไฟล์นั้นสามารถกำหนดตำแหน่งที่ต้องการอ่านหรือบันทึกข้อมูล คำสั่งที่ใช้ในการสร้างแรนดัมไฟล์ คือ คำสั่ง seek โดยมีลักษณะการเขียนดังที่เห็น

seek(fp,n);

คำสั่งดังกล่าว เป็นคำสั่งเลื่อนไฟล์พอยน์เตอร์ไปตำแหน่งที่ต้องการอ่านหรือบันทึก และตามด้วยคำสั่ง read(fp,ตัวแปร) หรือ write(fp,ตัวแปร)



แรนดัมไฟล์

สมมุติว่ามีข้อมูลอยู่ 100 ข้อมูล ลักษณะของซีเควนเชียลไฟล์ ไฟล์พอยน์เตอร์จะชี้ไปที่ข้อมูลแรกและเลื่อนไปทีละข้อมูล แต่สำหรับการอ่านหรือบันทึกข้อมูลแบบแรนดัมไฟล์ สามารถเลื่อนไฟล์พอยน์เตอร์ไปตำแหน่งที่ต้องการ โดยใช้คำสั่ง seek(fp,ตำแหน่งที่ต้องการ) ไฟล์พอยน์เตอร์ก็จะชี้ไปที่ข้อมูลที่ตำแหน่งที่ต้องการอ่านหรือบันทึกข้อมูล หลังจากนั้น ถ้าต้องอ่านข้อมูลก็ใช้คำสั่ง

read(fp,ตัวแปร)

หรือถ้าต้องการบันทึกหรือแก้ไขข้อมูลก็ใช้คำสั่ง

write(fp,ตัวแปร)

ตัวอย่างที่ 8.6 บันทึกข้อมูลลงในไฟล์โดยใส่ลำดับข้อมูลและข้อมูล จนกว่าลำดับข้อมูลเป็นลบ หลักจากนั้นอ่านข้อมูลโดยใส่ตำแหน่งข้อมูลที่ต้องการจะอ่าน

```
program randomfile;
type person=record
                 name:string[10];
                 year:integer
             end;
var fp:file of person;
   data:person;
   filename:string[12];
   rec:integer;
begin
  write('filename ? ');
  readln(filename);
  assign(fp,filename);
  rewrite(fp);                                   { open to write }
  readln(rec,data.year,data.name);
  while rec>0 do
     begin
       seek(fp,rec);           ............................... 1
       write(fp,data);                           { write to disk }
       readln(rec,data.year,data.name)
     end;
  reset(fp);
  write('rec ? '); readln(rec);
  while rec<>999 do           ............................... 2
     begin
       seek(fp,rec);
       read(fp,data);
       writeln(data.name:10,data.year:5);
       write('rec ? ');readln(rec)
     end;
  close(fp)
end.
```

```
filename ? b:random.dat  [enter]
1 21 Ann       [enter]
5 25 Nancy     [enter]
3 21 candy     [enter]
2 18 Rolla     [enter]
4 19 eluza     [enter]
0 -1 Nobody    [enter]
rec ? 1        [enter]
     Ann    21
```

1 กำหนดตำแหน่งข้อมูล โดยใช้คำสั่ง seek(fp,rec) และทำการบันทึกข้อมูลลงตามตำแหน่ง rec

2 กำหนดตำแหน่งข้อมูล และแสดงข้อมูลจนกว่าตำแหน่งข้อมูลเป็นลบ

8.5 เท็กซ์ไฟล์

< การดำเนินการข้อมูลแบบเท็กซ์ไฟล์ >

ไฟล์ที่ประกอบด้วยข้อมูลแบบตัวอักขระเรียกว่า เท็กซ์ไฟล์ โปรแกรมต่างๆที่อยู่ตั้งแต่บทแรกถึงบทสุดท้ายถือเป็นเท็กซ์ไฟล์ทั้งสิ้น

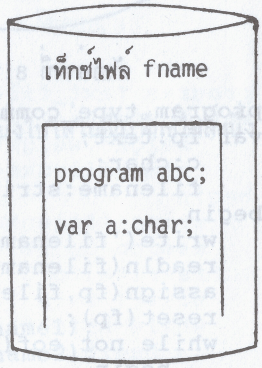
เท็กซ์ไฟล์ก็คือไฟล์ที่มีลักษณะเป็น

type text=file of char;

โดยปรกติแล้ว text เป็นแบบไฟล์ที่มีอยู่ในคอมไพเลอร์ ดังนั้น จึงไม่ต้องใช้คำสั่ง type กำหนดแบบข้างบนก็สามารถที่จะใช้ได้โดยตรง

สิ่งที่เท็กซ์ไฟล์แตกต่างจากไฟล์แบบอื่นคือ ท้ายของทุกๆบรรทัดในเท็กซ์ไฟล์มีรหัส CR (Carriage Return) อยู่ตอนท้ายของบรรทัด

การเปิดเท็กซ์ไฟล์เพื่ออ่านหรือบันทึกข้อมูลสามารถทำได้ดังนี้คือ




```
var fp:text;
assign(fp,fname); (* เปิดไฟล์เพื่ออ่าน  *)
reset(fp);
assign(fp,fname); (* เปิดไฟล์เพื่อบันทึก *)
rewrite(fp);
```

< ฟังก์ชันและโพรซิเจอร์ที่ใช้กับเท็กซ์ไฟล์ >

readln(ไฟล์พอยน์เตอร์,ตัวแปร)

... เหมือนคำสั่ง read แต่เมื่ออ่านข้อมูลแล้ว จะไปอ่านข้อมูลของต้นบรรทัดใหม่ ไม่ว่าบรรทัดที่อ่านนั้นจะมีข้อมูลเหลือหรือไม่

writeln(ไฟล์พอยน์เตอร์,ตัวแปร)

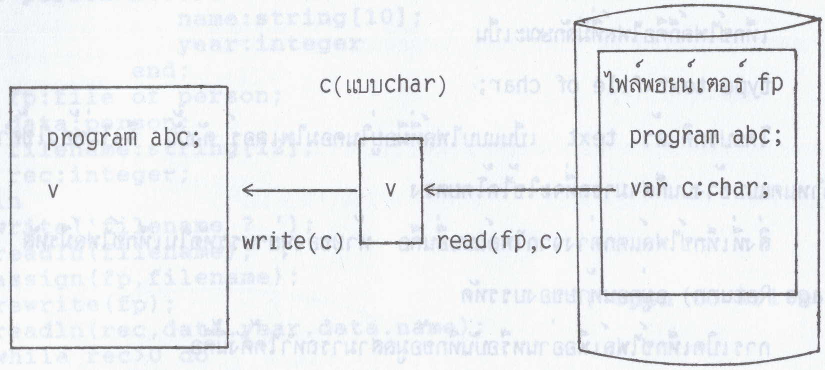
... เหมือนคำสั่ง write แต่เมื่อบันทึกข้อมูลแล้ว จะบันทึกรหัส CR ไปด้วย เพื่อเป็นการแสดงว่าจบการบันทึกในบรรทัดนั้นและขึ้นบรรทัดใหม่เพื่อทำการบันทึกข้อมูลต่อไป

eoln(ไฟล์พอยน์เตอร์) ฟังก์ชันแบบบูลีน

... ถ้าตำแหน่งของไฟล์พอยน์เตอร์อยู่สุดบรรทัด ค่าจะเป็น true ถ้าไม่ใช่ค่าจะเป็น false

< การอ่านเท็กซ์ไฟล์ >

เท็กซ์ไฟล์มีลักษณะเป็นไฟล์ของตัวอักขระ ถ้าต้องการ เรียกข้อมูลจากเท็กซ์ไฟล์และแสดงผลก็สามารถทำได้ดังนี้



ตัวอย่างที่ 8.7 โปรแกรมอ่านข้อมูลจากเท็กซ์ไฟล์ (ตัวอย่างที่ 8.1) และแสดงผล

```
program type_command;
var fp:text;                        ............................ 1
   c:char;
   filename:string[12];
begin
   write('filename ? ');
   readln(filename);                    .................... 2
   assign(fp,filename);
   reset(fp);                           .................... 3
   while not eof(fp) do                 .................... 4
     begin
        read(fp,c);                      { read from disk }
        write(c)
     end;
   close(fp)
end.
```

```
filename ? b:ex3_1.pas [enter]

program if_else;
var a:integer;
begin
   write('data ? ');
   readln(a);
   if a>=0 then
      writeln('plus')
   else
      writeln('minus')
end.
```

1 กำหนด fp ให้เป็นไฟล์พอยน์เตอร์ text

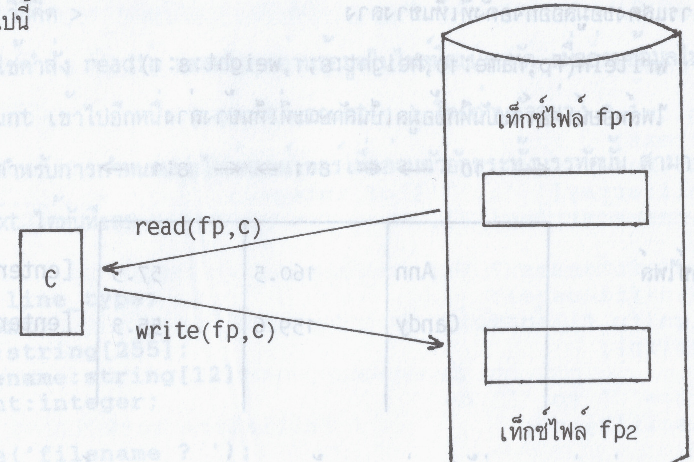
2 อ่านชื่อไฟล์ที่ต้องการจะเรียก

3 เปิดไฟล์เพื่อทำการอ่าน

4 อ่านข้อมูลจากไฟล์ตั้งแต่ต้นจนจบ

< การลอกข้อมูลจากไฟล์หนึ่งให้อีกไฟล์หนึ่ง >

สมมุติว่ามีเท็กซ์ไฟล์อยู่ไฟล์หนึ่ง ถ้าต้องการจะลอกข้อมูลมาใส่ไฟล์อีกไฟล์หนึ่งก็สามารถทำได้ดังต่อไปนี้



ตัวอย่างที่ 8.8 โปรแกรมการลอกข้อมูลจากเท็กซ์ไฟล์หนึ่งไปใส่ในอีกเท็กซ์ไฟล์หนึ่ง

```
program copy_command;
var fp1,fp2:text;
    c:char;
    fname1,fname2:string[12];
begin
  write('source file ? ');       readln(fname1);
  write('destination file ? '); readln(fname2);
  assign(fp1,fname1);
  reset(fp1);                     { open to read }
  assign(fp2,fname2);
  rewrite(fp2);                   { open to write }
  while not eof(fp1) do
    begin
      read(fp1,c);                { read from disk to c } .. 1
      write(fp2,c)                { write c to disk }     .. 2
    end;
  close(fp1);
  close(fp2)
end.

source file ?       b:ex3_2.pas     [enter]
destination file ? b:newex3_2.pas    [enter]
```

1 อ่านไฟล์แรกโดยไฟล์พอยน์เตอร์ fp1 ให้กับ C โดยคำสั่ง read(fp1,c)

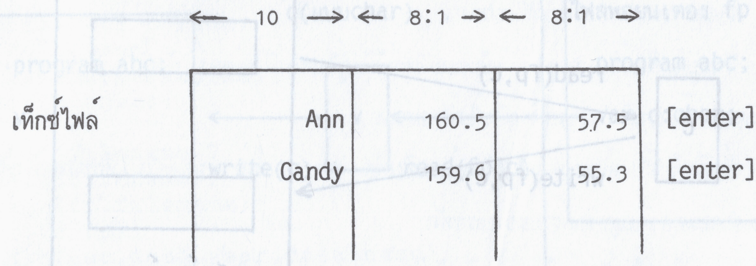
2 บันทึกข้อมูลที่อ่านได้โดยคำสั่ง write(fp2,c) ดำเนินการไปจนกระทั่งจบไฟล์

< การกำหนดแบบการบันทึกข้อมูลแบบเท็กซ์ >

คำสั่งที่ใช้ในการบันทึกข้อมูลแบบเท็กซ์ไฟล์ (write/writeln) สามารถกำหนดแบบได้เหมือนการแสดงข้อมูลออกจอดังที่เห็นข้างล่าง

```
writeln(fp,name:10,height:8:1,weight:8:1);
```

ไฟล์พอยน์เตอร์จะบันทึกข้อมูลเป็นลักษณะที่เห็นข้างล่าง



ตัวอย่างที่ 8.9 ใส่ข้อมูล ส่วนสูง น้ำหนักและชื่อ บันทึกลงไฟล์ และแสดงข้อมูลออกจอ

```
program text_file;
var fp:text;
   height,weight:real;
   name,filename:string[12];
begin
  write('filename ? ');readln(filename);
  assign(fp,filename);
  rewrite(fp);
  write('Data ? ');readln(height,weight,name);
  while height>0 do                                    ......... 1
     begin
       writeln(fp,name:10,height:8:1,weight:8:1);
       write('Data ? ');readln(height,weight,name)
     end;
  reset(fp);
  while not eof(fp) do
     begin
       readln(fp,name,height,weight);
       writeln(name:10,height:8:1,weight:8:1) ......... 2
     end;
  close(fp)
end.
```

```
filename ? b:person.dat          [enter]
Data ? 160.5 57.5 Ann            [enter]
Data ? 159.6 55.3 Candy          [enter]
Data ? -1 1 /                    [enter]

      Ann        160.5      57.5
    Candy        159.6      55.3
```

1 บันทึกข้อมูลจนกว่า height เป็นลบ

2 แสดงข้อมูลตั้งแต่ต้นจนจบไฟล์ โดยอ่านฟิลด์แรกให้กับตัวแปรแบบ string[10] อีก 2 ฟิลด์ให้กับตัวแปรแบบ real

ตัวอย่างที่ 8.10 อ่านข้อมูลจากเท็กซ์ไฟล์เช่นเดียวกับตัวอย่างที่ 8.8 และแสดงข้อมูลและแสดงตัวเลขแสดงว่าปัจจุบันอยู่ที่บรรทัดที่เท่าใด

< วิธีคิด >

ใช้คำสั่ง readln และทำการอ่านข้อมูลในไฟล์ทีละบรรทัด เมื่ออ่านข้อมูลไปบรรทัดหนึ่งก็เพิ่มค่า count เข้าไปอีกหนึ่ง (count:=count+1) โดยอ่านตัวอย่าง 8.1

สำหรับการกำหนดแบบไฟล์พอยน์เตอร์เพื่ออ่านตัวอักขระทั้งบรรทัดนั้น สามารถกำหนดให้เป็นแบบ text ได้ทันทีเลย

```
program line_type;
var f:text;
    buf:string[255];
    filename:string[12];
    count:integer;
begin
   write('filename ? ');
   readln(filename);
   assign(f,filename);
   reset(f);
   count:=1;
   while not eof(f) do
      begin
        readln(f,buf);               ........ 1
        writeln(count:4,': ',buf);   ........ 2
        count:=count+1               ........ 3
      end;
   close(f)
end.
```

```
filename ? b:rei3_1.pas  [enter]

   1: program if_else;
   2: var a:integer;
   3: begin
   4:    write('data ? ');
   5:    readln(a);
   6:    if a>=0 then
   7:       writeln('plus')
   8:    else
   9:       writeln('minus')
  10: end.
```

1 อ่านข้อมูลทีละบรรทัดและเก็บไว้ที่ buf

2 แสดงบรรทัด และข้อมูล

3 เพิ่มค่า count เข้าไปอีกหนึ่ง

ตัวอย่างที่ 8.11 อ่านข้อมูลจากไฟล์ (ตัวอย่างที่ 8.1) และแสดงว่ามีตัวอักขระต่างๆ อยู่เท่าใด

< วิธีคิด > ให้ ascii เป็นตัวแปรอาร์เรย์มีอินเด็กซ์ตั้งแต่ '_'..'~' แบบเลขจำนวนเต็ม เมื่ออ่านข้อมูลให้ c แล้วก็ให้เพิ่มค่า ascii[c] จากค่าเก่าเข้าไปอีก 1

```
program char_count;
var fp:text;
   c,k:char;
   ascii:array[' '..'˜'] of integer;                    .......... 1
   filename:string[12];
begin
   write('filename ? ');
   readln(filename);
   assign(fp,filename);
   reset(fp);

   for k:=' ' to '˜' do
      ascii[k]:=0;                 { initialize to 0 }

   while not eof(fp) do  { start counting}
       begin
          read(fp,c);
          if c in [' '..'˜'] then ascii[c]:=ascii[c]+1   .. 2
       end;

   for k:=' ' to '˜' do
     if ord(k) mod 6=1 then
        writeln(k:4,'=':4,ascii[k]:4)
     else write(k:4,'=':4,ascii[k]:4)
end.
```

```
filename ? b:ex8_1.pas
    = 87  !  =  0  "  =  0  #  =  0  $  =  0  %  =  0
 &  =  0  '  =  2  (  =  8  )  =  8  *  =  0  +  =  0
 ,  =  2  -  =  2  .  =  2  /  =  0  0  =  2  1  =  2
 2  =  0  3  =  0  4  =  0  5  =  4  6  =  1  7  =  1
 8  =  1  9  =  6  :  =  4  ;  = 11  <  =  1  =  =  0
 >  =  1  ?  =  2  @  =  0  A  =  0  B  =  0  C  =  0
 D  =  0  E  =  0  F  =  0  G  =  0  H  =  0  I  =  0
 J  =  0  K  =  0  L  =  0  M  =  0  N  =  0  O  =  0
 P  =  0  Q  =  0  R  =  0  S  =  0  T  =  0  U  =  0
 V  =  0  W  =  0  X  =  0  Y  =  0  Z  =  0  [  =  7
 \  =  0  ]  =  7  ^  =  0  _  =  0  `  =  0  a  = 11
 b  =  3  c  =  1  d  =  7  e  = 42  f  = 12  g  =  7
 h  =  1  i  = 22  j  =  0  k  =  1  l  = 10  m  =  7
 n  = 22  o  =  4  p  =  6  q  =  0  r  = 19  s  =  7
 t  = 16  u  =  1  v  =  1  w  =  4  x  =  0  y  =  0
 z  =  0  {  =  0  |  =  0  }  =  0  ˜  =  0
```

1 กำหนดให้ ascii เป็น array['_'..'˜'] of integer;

2 อ่านข้อมูลจากไฟล์ให้ c และเพิ่มค่า ascii[c] จากเดิมอีก 1

ตัวอย่างที่ 8.12 hashing function

จากตัวอย่างที่ 8.6 ในกรณีที่ต้องการเรียกข้อมูลจากแรนดัมไฟล์ จำเป็นต้องบอกลำดับข้อมูล ถ้าหากว่าไม่ทราบถึงลำดับข้อมูลแต่ทราบชื่อของบุคคล ก็สามารถใช้วิธี hashing หาข้อมูลได้ hashing เป็นวิธีการเลือกเอาอักขระบางตัวจากข้อมูลชื่อมาแปลงเป็นตัวเลข

< วิธีคิด >

hashing function มีหลายวิธี ในที่นี้จะยกมาเพียงวิธีเดียว

สมมุติว่ามีข้อมูลที่เป็นชื่อภาษาอังกฤษ A_1 A_2...A_n (n ตัว) หลักการของ hashing ก็คือ เลือกตัวแรก (A_1) ตัวกลาง ($A_{n/2}$) และตัวสุดท้าย (A_n) มาแปลงเป็นตัวเลข

การคำนวณจะใช้ฟังก์ชัน hashing function ซึ่งมีค่าเป็น

$hash(A_1,A_2,A_3)=(A_1+A_{n/2}*26+A_n*26*26) \bmod 1000$

ยกตัวอย่างเช่น

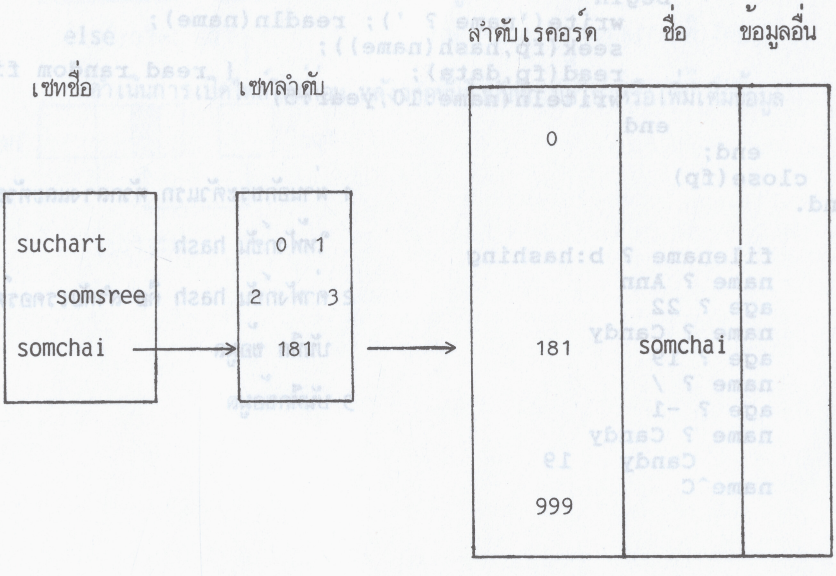
hash('somchai')=

((ord('s')-96)+(ord('c')-96)*26+(ord('i')-96)*26*26) mod 1000

=((116-96)+(100-96)*26+(105-96)*26*26) mod 1000

=(19+78+6084) mod 1000 = 181

ค่า 181 ก็คือ ตำแหน่งเรคอร์ดที่เก็บข้อมูลของ somchai สำหรับเลข 96 คือ รหัส ascii ก่อน 'a' หนึ่งตัว



```
program hashing;
type nametag=string[12];
     person=record
               name:nametag;
               year:integer
            end;
var fp:file of person;
    data:person;
    filename:nametag;

    function hash(a:nametag):integer;               ............... 1
      var n:integer;
      begin
        n:=length(a);      { length is string's length }
        hash:=((ord(a[1])-96)+(ord(a[n div 2])-96)*26
               +(ord(a[n])-96)*26*26) mod 1000
      end;

begin
   write('filename ? '); readln(filename);
   assign(fp,filename);  rewrite(fp);    { open to write }
   with data do
     begin
       write('name ? '); readln(name);
       write('age ? ');  readln(year);
       while year>0 do
         begin
           seek(fp,hash(name));                   ............... 2
           write(fp,data);                   { random write } .. 3
           write('name ? '); readln(name);
           write('age ? ');  readln(year)
         end;
       reset(fp);                            { open to read }
       while true do
         begin
           write('name ? '); readln(name);
           seek(fp,hash(name));
           read(fp,data);                    { read random file }
           writeln(name:10,year:5)
         end
     end;
  close(fp)
end.

    filename ? b:hashing
    name ? Ann
    age ? 22
    name ? Candy
    age ? 19
    name ? /
    age ? -1
    name ? Candy
         Candy   19
    name^C
```

1 ผ่านอักขระตัวแรก ตัวกลางและตัวสุดท้ายให้ฟังก์ชัน hash

2 ค่าฟังก์ชัน hash คือ ลำดับเรคอร์ดที่จะบันทึก ข้อมูล

3 บันทึกข้อมูล

< ฟังก์ชัน ioresult >

ioresult สามารถเขียนเป็นลักษณะ

ตัวแปร (แบบบูลีน) := (ioresult=0);

ioresult จะส่งค่า 0 กลับมาในกรณีที่ไม่มี I/O error แต่ในกรณีที่มี ioresult=0 ก็จะเป็น false

ข้อควรระวังสำหรับคำสั่งนี้ก็คือ ioresult เป็นฟังก์ชันตรวจว่ามี I/O error หรือไม่ ในกรณีที่เกิด I/O error การดำเนินการก็จะหยุดทันที ทำให้ไม่สามารถนำโปรแกรมไปใช้งานได้ วิธีการป้องกันไม่ให้โปรแกรมหยุดการดำเนินการก็คือ ใส่ {$I-} หรือ (*$I-*) เมื่อกำหนดให้ {$I-} หรือ (*$I*) แล้ว ถึงแม้จะเกิด I/O error โปรแกรมก็ยังดำเนินการต่อไป โดยปรกติใช้กับคำสั่ง reset เนื่องจากคำสั่ง reset จะเกิด I/O error ทันที ในกรณีที่ไม่มีไฟล์ที่ต้องการอ่านข้อมูลอยู่ เมื่อใช้ {$I-} แล้วควรใช้ {$I+} ควบคู่กันไปด้วย

```
ex    assign(ไฟล์พอยน์เตอร์,ชื่อไฟล์);
      {$I-}
      reset(ไฟล์พอยน์เตอร์);
      {$I+}
      if (check:=(ioresult=0)) then
         ดำเนินการอ่าน แก้ไข หรือเพิ่มเติมข้อมูล
        else
         ดำเนินการเปิดไฟล์ใหม่ก่อน หลังจากนั้นจึงบันทึก แก้ไข หรือเพิ่มเติมข้อมูล
```

Coffee Break

ใน Standard Pascal คำสั่งที่ใช้ในการดำเนินการจัดการไฟล์คือ get กับ put ซึ่งไม่มีใน Turbo Pascal ตัวอย่างข้างล่างเป็นโปรแกรมลอกข้อมูลจากไฟล์หนึ่งไปยังอีกไฟล์หนึ่ง (เหมือนตัวอย่างที่ 8.5)

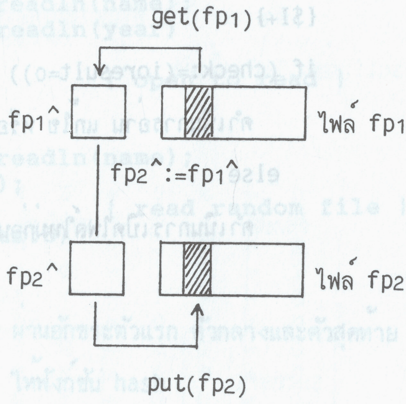
```
program filecopy(fp1,fp2);
var fp1,fp2:file of char;
begin
   reset(fp1);
   rewrite(fp2);
   while not eof(fp1) do
      begin
         fp2^:=fp1^;
         put(fp2);
         get(fp1)
      end
end.
```

reset(fp_1) คือ คำสั่งเปิดไฟล์ fp_1 เพื่ออ่านโดยบัฟเฟอร์ fp_1^

rewrite(fp_2) คือ คำสั่งเปิดไฟล์ fp_2 เพื่อบันทึกโดยบัฟเฟอร์ fp_2^

get คือการอ่านข้อมูลแบบบัฟเฟอร์

put คือการบันทึกข้อมูลแบบบัฟเฟอร์



แบบฝึกหัด

1 จงเขียนโปรแกรมเก็บข้อมูลนักเรียนกลุ่มหนึ่งซึ่งมีรายละเอียดดังนี้ ชื่อ อายุ ชั้นเรียน และที่อยู่ โดยทำเป็นเมนูเลือกได้ว่าต้องการดำเนินการดังที่เห็น

ก) แสดงข้อมูลทุกคน

ข) แสดงข้อมูลนักศึกษาคนเดียว

ค) เริ่มใส่ข้อมูล

ง) เพิ่มเติมข้อมูล

จ) แก้ไขข้อมูล

ง) เสร็จงาน

2 จงเขียนโปรแกรมเก็บข้อมูลเลขโทรศัพท์ลงดิสก์ โดยสามารถแก้ไขและเพิ่มเติมข้อมูลได้

3 จงเขียนโปรแกรมเก็บข้อมูลของห้องพัก โดยสามารถทำการใส่เบอร์ห้อง ชื่อที่ผู้มาพัก วันที่เข้ามาพักและวันออก และสามารถตรวจว่าระหว่างวันที่เท่าใดถึงวันที่เท่าใดมีห้องใดว่างหรือไม่

บทที่ 9

โพรซิเจอร์และฟังก์ชัน

9.1 โพรซีเจอร์และฟังก์ชันมาตรฐาน

โพรซิเจอร์และฟังก์ชันมาตรฐานเป็นส่วนที่ได้สร้างไว้ใน system เรียบร้อยแล้ว ซึ่งสามารถเรียกมาใช้ได้โดยไม่ต้องสร้างเอง

ที่ผ่านมาได้มีการใช้โพรซิเจอร์มาตรฐานไปบ้างแล้ว เช่น read write เป็นต้น ส่วนฟังก์ชันมาตรฐานก็ได้แก่ eof (end of file) ต่อไปจะกล่าวถึงโพรซิเจอร์และฟังก์ชันอื่นๆ ในภาษา Pascal อนึ่ง * คือ ฟังก์ชันหรือโพรซิเจอร์ที่สามารถใช้ใน Standard Pascal

ประเภท	ชื่อโพรซิเจอร์	บทที่
อินพุทเอาพุท	* read * readln * write * writeln	2
จัดการไฟล์	assign * close erase rename * reset * rewrite seek * read * read * readln * write * writeln	8
จัดการสตริง	delete insert str val	9
หน่วยความจำ	getmem mark * new release * dispose	10

โพรซิเจอร์มาตรฐาน

หมายเหตุ การดำเนินการไฟล์ใน Standard Pascal ต้องใช้คำสั่ง get/put

ประเภท	ชื่อฟังก์ชัน	บทที่
การคำนวณ	* abs * arctan * cos * exp frac * int * sin * sqrt * round * trunc random	9
สเกลาร์	odd * pred * succ * chr * ord	9
ไฟล์	* eof * eoln filepos filesize	8
สตริง	concat copy length pos	9

ฟังก์ชันมาตรฐาน

9.2 ฟังก์ชันที่ใช้ในการคำนวณ

ในภาษา Pascal ฟังก์ชันที่ใช้ในการคำนวณมีดังนี้คือ

ฟังก์ชัน	การดำเนินการ	แบบพารามิเตอร์	แบบฟังก์ชัน
abs(x)	ส่งค่าสัมบูรณ์ของ x	เลขจำนวนเต็ม หรือ เลขจำนวนจริง	เหมือนพารามิเตอร์
sqr(x)	ส่งค่ายกกำลังสองของ x		
sin(x)	ส่งค่า sin ของ x	เลขจำนวนเต็ม หรือ เลขจำนวนจริง	เลขจำนวนจริง
cos(x)	ส่งค่า cos ของ x		
arctan(x)	ส่งค่า arctan ของ x		
exp(x)	ส่งค่า e ยกกำลัง x		
ln(x)	ส่งค่า LOGe(x)		
sqrt(x)	ส่งค่ารากที่สองของ x		
frac(x)	ส่งค่าหลังจุดทศนิยมของ x		
int(x)	ส่งค่าก่อนจุดทศนิยมของ x		
trunc(x)	ปัดค่าหลังจุดทศนิยมของ x	เลขจำนวนจริง	เลขจำนวนเต็ม
round(x)	ปัดค่าหลังจุดทศนิยมถ้าน้อยกว่า 5 หรือปัดขึ้นถ้ามากกว่า 5		
randomize	เริ่มต้นดำเนินการสุ่มตัวเลข	ไม่มี	ไม่มี
random	เลขสุ่มระหว่าง 0 ถึง 1	ไม่มี	เลขจำนวนจริง
random(x)	เลขสุ่มระหว่าง 0 ถึง x	เลขจำนวนเต็ม	เลขจำนวนเต็ม
pi	ส่งค่า	ไม่มี	เลขจำนวนจริง

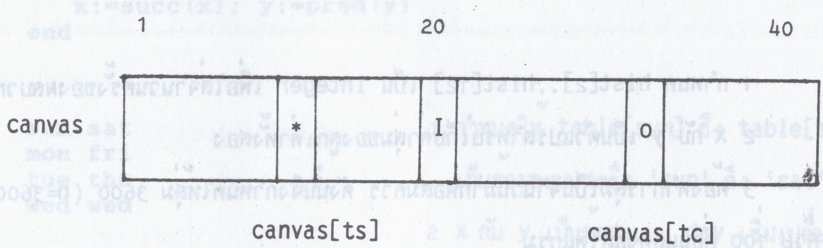
ตัวอย่างที่ 9.1 การเขียนกราฟ sin บนจอโดยใช้ * และกราฟ cos โดยใช้ o

```
program sin_cos;
var n,ts,tc:integer;
   rd:real;
   canvas:string[40];
begin
   rd:=pi/180;
   writeln('    -1.........+.........0.........+.........1');
   for n:=1 to 18 do                                            1
      begin
         canvas:='                    I                   ';
         ts:=21+round(20*sin(n*20*rd));
         tc:=21+round(20*cos(n*20*rd));
         canvas[ts]:='*'; canvas[tc]:='o';                      2
         writeln(n*20:4,' ',canvas)
      end
end.
```

```
    -1.........+.........0.........+.........1
  20                     I      *           o
  40                     I            * o
  60                     I         o      *
  80                     I  o                *
 100                  o  I                   *
 120           o         I                *
 140      o              I            *
 160  o                  I      *
 180 o                   I*
 200  o           *      I
 220      o *            I
 240    *      o         I
 260 *                o  I
 280 *                   I  o
 300    *                I         o
 320        *            I              o
 340              *      I                  o
 360                     I*                  o
```

1 ใช้คำสั่งวนซ้ำ for วนตั้งแต่ 0 ถึง 18 (เมื่อคูณ 20 ก็จะได้เป็นมุม 0 ถึง 360)

2 สร้างสตริง canvas โดยให้ I อยู่ตรงกลาง ใส่ '*' เมื่อคำนวณค่า sin(ts)ได้ และใส่ 'o' เมื่อคำนวณค่า cos (tc) ได้



3 แสดง canvas ทั้งบรรทัด

ตัวอย่างที่ 9.2 การแสดงฮีสโตแกรมของผลบวกของลูกเต๋าสองลูก โดยการสุ่มตัวเลขตั้งแต่ 1 ถึง 6 ให้กับตัวแปรสองตัวแปร และเก็บจำนวนครั้งของผลบวกที่ hist[2]..hist[12] และแสดงผลเมื่อดำเนินการสุ่มไปเป็นจำนวนมากพอ

```
program two_dice;
const n=3600;
var i,j,k:integer;
    hist:array[2..12] of integer;          ........... 1
begin
   for i:=2 to 12 do
      hist[i]:=0;
   for i:=1 to n do
      begin
         j:=random(6)+1;
         k:=random(6)+1;                     ........... 2
         hist[j+k]:=hist[j+k]+1
      end;
   for i:=2 to 12 do
      begin
         write(i:2,'  :  ');
         for j:=1 to hist[i] div 20 do      .......... 3
            write('*');
         writeln
      end
end.
```

```
     2  :  ****
     3  :  *********
     4  :  *************
     5  :  **********************
     6  :  ************************
     7  :  ***************************
     8  :  *************************
     9  :  *******************
    10  :  ****************
    11  :  **********
    12  :  *****
```

1 กำหนด hist[2]..hist[12] เป็น integer เพื่อใส่จำนวนครั้งของผลบวกลูกเต๋า

2 x กับ y เป็นตัวแปรสำหรับเก็บค่าสุ่มของลูกเต๋าทั้งสอง

3 ต้องทำการสุ่มเป็นจำนวนมากพอสมควร ดังนั้นจึงกำหนดให้สุ่ม 3600 (n=3600) และหารด้วย 100 ก่อนแสดงฮีสโตแกรม

๙.๓ ฟังก์ชันของข้อมูลแบบสเกลาร์

ข้อมูลแบบสเกลาร์มีฟังก์ชันมาตรฐานที่สามารถใช้ได้ดังต่อไปนี้คือ

ฟังก์ชันมาตราฐาน	การดำเนินการ	แบบพารามิเตอร์	แบบฟังก์ชัน
ord(x)	ส่งลำดับของข้อมูล x	สเกลาร์,อักขระ	เลขจำนวนเต็ม
pred(x)	ส่งข้อมูลก่อน x	สเกลาร์	สเกลาร์
succ(x)	ส่งข้อมูลหลัง x	สเกลาร์	สเกลาร์
chr(x)	ส่งรหัส ascii ของ x	เลขจำนวนเต็ม	อักขระ
odd(x)	ส่ง true ถ้า x เป็นเลขคี่ ส่ง false ถ้า x เป็นเลขคู่	สเกลาร์	บูลีน

ex type day=(sun,mon,tue,wed,thu,fri,sat);

succ(sun) คือ mor pred(mon) คือ sun

ord(tue) คือ 2 (ตัวแรกคือ 0) pred(sun) -------> error

ตัวอย่างที่ 9.3 แสดงค่าข้อมูลแบบ day โดยใช้คำสั่ง succ กับ pred

```
program scalar;
type day=(sun,mon,tue,wed,thu,fri,sat);
var x,y:day;
    table:array[sun..sat] of string[3];
begin
   table[sun]:='sun';table[mon]:='mon';table[tue]:='tue'; ..... 1
   table[wed]:='wed';table[thu]:='thu';table[fri]:='fri';
   table[sat]:='sat';
   x:=sun; y:=sat;
   while x<=y do                   ............................... 2
      begin
         writeln(table[x]:4,table[y]:4);
         x:=succ(x); y:=pred(y)
      end
end.
```

```
sun sat
mon fri
tue thu
wed wed
```

1 กำหนดให้ table[sun] ถึง table[sat] เก็บข้อมูลแบบสตริง 'sun' ถึง 'sat'

2 x กับ y เป็นข้อมูลแบบ day เริ่มเปลี่ยน x จาก sun ลง และเปลี่ยน y จาก sat ขึ้น

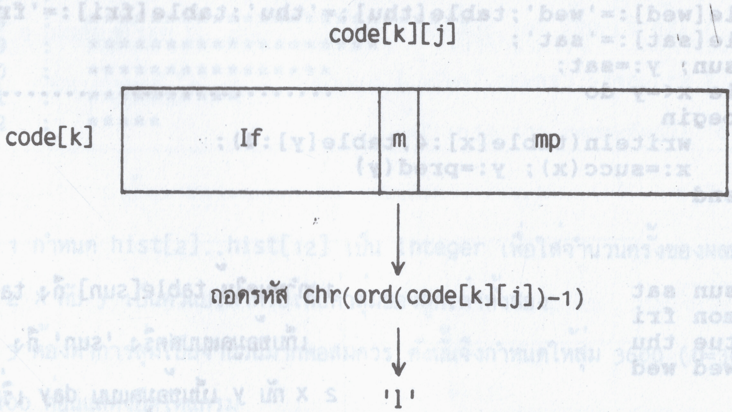
ตัวอย่างที่ 9.4 การถอดรหัสข้อความชุดหนึ่งด้วยการอ่านข้อมูลและถอดรหัส โดยวิธีแสดงตัวอักษรตัวต่อไปเช่น ถ้าอ่าน 'A' ก็ถอดรหัสเป็น 'B'

```
program decode;
const n=2;
var code:array[1..n] of string[50];        .................. 1
    j,k:integer;
begin
   code[1]:='Ifmmp';
   code[2]:='Uvscp!Qbtdbm';
   for k:=1 to n do
      begin
         for j:=1 to length(code[k]) do
            write(chr(ord(code[k][j])-1));  .................. 2
         writeln
      end
end.
```

```
Hello
Turbo Pascal
```

1 ให้ code[1] และ code[2] เป็นข้อมูล string[50] เพื่อเก็บข้อมูลที่ยังไม่ถอดรหัส

2 code[k][j] หมายถึง อักษรตัวที่ j ของรหัส code[k] เมื่อต้องการถอดรหัส ต้องหาลำดับ (ord(code[k][j])) และลบด้วย 1 หลังจากนั้นก็ใช้ฟังก์ชัน chr เพื่อเรียกอักษรดังที่เห็นข้างล่าง



9.4 โพรซิเจอร์และฟังก์ชันมาตรฐานที่ใช้กับสตริง

โพรซิเจอร์และฟังก์ชันที่จะกล่าวต่อไปนี้ไม่มีใน Standard Pascal เนื่องจากใน Standard Pascal ไม่มีข้อมูลแบบสตริง โพรซิเจอร์ที่ใช้กับข้อมูลแบบสตริงมีดังนี้คือ

< delete > ตัดข้อความบางส่วน

แบบ delete(st,pos,n)

การดำเนินการ

ลบสตริง st หลังตำแหน่ง pos ไป n ตัว (pos กับ n ต้องเป็น integer)

ex st:='Hello';

delete(st,2,3) ..... st จะมีค่าเป็น 'He'

< insert > เติมข้อความ

แบบ insert(st1,st2,pos)

การดำเนินการ

เติม st1 ลงใน st2 หน้าตำแหน่ง pos

ex st2:='Helloworld;

insert('_Pascal_',st2,6);

... st2 จะมีค่าเป็น 'Hello_Pascal_world';

< str > แปลงตัวเลขเป็นสตริง

แบบ str(x:ความกว้าง,st)

การดำเนินการ

x เป็นตัวแปรแบบตัวเลข เมื่อเรียกโพรซิเจอร์ str แล้ว ตัวแปร st จะเป็นสตริงที่มีรายละเอียดเท่ากับ x (แต่ตัวแปร st เป็นสตริงจึงไม่มีค่าเหมือน x) สำหรับความกว้างนั้นกำหนดได้เหมือนคำสั่ง write เช่น n:5 หรือ f:6:2 เป็นต้น

ex x:=123; str(x:5,st);

...st จะมีค่าเป็น ' 123'

y:=10.5; str(y:5:1,st)

...st จะมีค่าเป็น ' 10.5'

< val > แปลงจากสตริงเป็นตัวเลข (ตรงข้ามกับ str)

แบบ val(st,x,flag)

การดำเนินการ

แปลงค่าสตริงของ st เป็นตัวเลขและให้ค่าแก่ x ในกรณีที่ไม่มี error กล่าวคือค่าสตริง st สามารถแปลงเป็นค่าตัวเลขได้ flag ก็จะเป็น 0 ในกรณีที่ไม่สามารถแปลงได้จะเกิด error ค่า flag ก็คือตำแหน่งของสตริง st ที่เกิด error สำหรับ x อาจเป็นเลขจำนวนเต็มหรือจำนวนจริง ส่วน flag นั้นเป็นเลขจำนวนเต็ม

ฟังก์ชันที่ใช้กับข้อมูลแบบสตริงมีดังนี้คือ

< copy > ส่งข้อมูลบางส่วน

แบบ copy(st,pos,n)

การดำเนินการ

ส่งค่าสตริง st จากตำแหน่ง pos ไป n ตัว

ex st:='Pascal';

b:=copy(st,2,3); .... b จะมีค่าเป็น 'asc'

< length > ส่งความยาวของข้อมูลในสตริง

แบบ length(st)

การดำเนินการ

ส่งความยาวของสตริง st

ex length('Hello'); ... ส่งค่า 5 (เท่ากับ st[0])

< pos > ส่งตำแหน่งอักขระในสตริง

แบบ pos(key,st)

การดำเนินการ

หาตำแหน่ง key ในสตริง st และส่งตำแหน่งที่ตรวจได้กลับ ในกรณีที่ไม่พบจะส่ง 0

ex st:='Pascal';

pos('al',st) .... ส่งค่า 5

pos('h',st) .... ส่งค่า 0

ตัวอย่างที่ 9.5 โปรแกรมเปลี่ยนข้อความบางส่วนในสตริง

```
program replace;
var sentence,key,rep:string[255];
    n:integer;
begin
   sentence:='My name is Candy.I am 18 years old';
   key:='Candy';
   rep:='Ann';
   n:=pos(key,sentence);               ............. 1
   delete(sentence,n,length(key));     ............. 2
   insert(rep,sentence,n);             ............. 3
   writeln(sentence)
end.
```

```
My name is Ann.I am 18 years old
```

1 ให้ n มีค่าเท่ากับตำแหน่งของ 'Candy' ของสตริง sentence

2 ลบ 'Candy' ออกจาก sentence

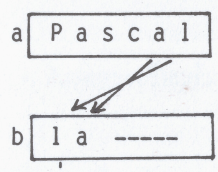
3 ใส่ค่า Ann ในตำแหน่งที่ Candy เคยอยู่

ตัวอย่างที่ 9.6 การแสดงค่าสตริง 'Pascal' และสตริงที่ส่งอักษรกลับข้างทีละอักษร

```
program string_copy;
type str=string[80];
var a,b:str;
    n,j:integer;
    procedure reverse(var a,b:str);    ............ 1
      var i:integer;
      begin
        b:='';
        for i:=length(a) downto 1 do
           b:=b+a[i]
      end;
begin
  a:='Pascal';
  reverse(a,b);
  n:=length(a);
  for j:=1 to n do
     begin
       write(copy(a,n-j+1,j):n);        ............ 2
       writeln(copy(b,1,j))             ............ 3
     end
end.
```

```
         alla
        callac
       scallacs
      ascallacsa
     PascallacsaP
```

1 ให้ b เป็นสตริงกลับข้างกับ a



2 แสดงค่า a จากทางขวา โดยเพิ่มทีละตัว

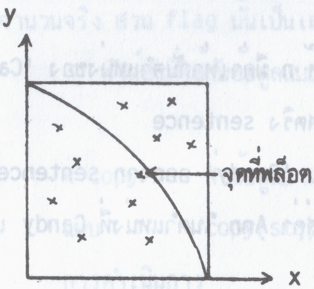
3 แสดงค่า b จากทางซ้าย โดยเพิ่มทีละตัว

ตัวอย่างที่ 9.6 การหาค่า π โดยวิธี Monte Carlo

วิธีคิด

สร้างเลขสุ่มโดยใช้คำสั่ง random ตั้งแต่ 0 ถึง 1 ขึ้นมา และให้ค่ากับตัวแปร x,y และกำหนดให้ค่า x y ที่ได้นั้นพล็อตลงบนจุดในสี่เหลี่ยมที่มีขนาด 1*1

ดังนั้น อัตรส่วนระหว่างพื้นที่สี่เหลี่ยมกับ 1/4ของพื้นที่วงกลม จะเท่ากับอัตรส่วนของจำนวนจุดที่อยู่ใน 1/4 ของวงกลมกับจำนวนจุดทั้งหมด



ถ้า a เป็นจำนวนจุดภายใน 1/4 ของวงกลม

ถ้า b เป็นจำนวนจุดภายนอก 1/4 ของวงกลม

เพราะฉะนั้นจะได้ $\pi/4:1 = a:(a+b)$

หรือ $\pi = a*4/(a+b)$

```
program test_pai;
var count,x,y,pai:real;
   n,k:integer;
begin
   writeln('n':5,'pai':10);
   for n:=1 to 5 do
      begin
         count:=0;
         for k:=1 to n*1000 do                          ......... 1
            begin
               x:=random;y:=random;
               if x*x+y*y<1 then count:=count+1        ......... 2
            end;
         pai:=4*count/(n*1000);                         ......... 3
         writeln(n*1000:5,pai:10:6)
      end
end.
```

1 สุ่มตัวเลขตั้งแต่ 1000,2000,...5000

2 ถ้าอยู่ใน 1/4 ของวงกลมให้เพิ่ม count เข้าไปอีกหนึ่ง

3 คำนวณค่า π

แบบฝึกหัด

1 จงสร้างตาราง sin cos tan ของมุมตั้งแต่ 0 ถึง 90 องศาโดยเปลี่ยนทีละ 10 องศา

2 จงเขียนโปรแกรมเพื่อให้ได้เอาท์พุทเป็น

```
.5
.25
.125
.0625
.03125
.015625
.0078125
.00390625
.001953125
.0009765625
```

บทที่ 10

โครงสร้างข้อมูลและอาร์กอริธึม

10.1 การเรียกตัวเอง (Recursion)

< โพรซิเจอร์เรียกตัวเอง >

การเรียกตัวเองคือ การคำนวณค่าลำดับปัจจุบัน (สมมุติว่าลำดับที่ n)โดยเรียกค่าของลำดับที่แล้ว (ลำดับที่ n-1) ลำดับที่ n-1 ก็ได้มาจากการเรียกลำดับที่ n-2 จะเห็นได้ว่าเป็นการเรียกโดยไม่สิ้นสุด ดังนั้น บรกติจึงมีการกำหนดค่าที่ตัวใดตัวหนึ่ง (เช่นลำดับที่ 0) เมื่อมีการเรียกถึงลำดับนี้ ก็จะหยุดการเรียกตัวเอง

แนวคิดของการเรียกตัวเองนี้ สามารถนำไปใช้กับการสร้างโพรซิเจอร์ได้เช่นกัน ซึ่งเรียกว่าโพรซิเจอร์เรียกตัวเอง หรือ Recusive Procedure

การทำความเข้าใจหลักการเรียกตัวเองอาจจะยากในตอนแรก แต่ถ้าทำความเคยชินแล้วก็นับว่าไม่ยากนัก ตัวอย่าง การคำนวณค่าแฟคตอเรี่ยล (n!) นั้นสามารถคำนวณโดยใช้สูตร

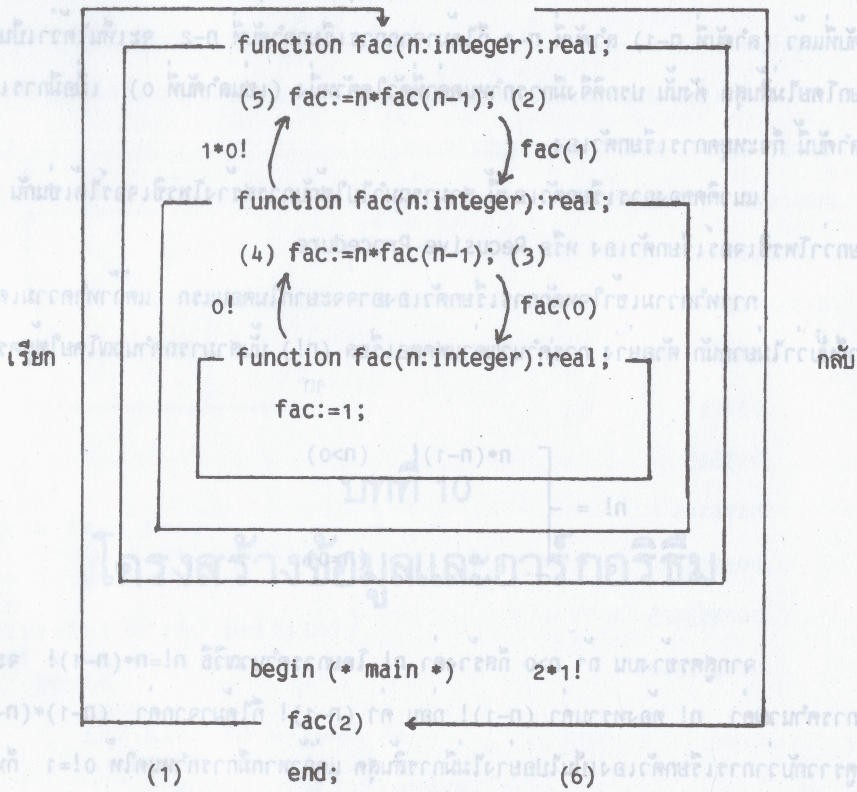
$$n! = \begin{cases} n*(n-1)! & (n>0) \\ 1 & (n=0) \end{cases}$$

จากสูตรข้างบน ถ้า n>0 ก็สร้างค่า n! โดยการคำนวณวิธี n!=n*(n-1)! จะเห็นว่าการคำนวณค่า n! ต้องทราบค่า (n-1)! ก่อน ค่า (n-1)! ก็ได้มาจากค่า (n-1)*(n-2)! ซึ่งดูราวกับว่าการเรียกตัวเองเป็นไปอย่างไม่มีการสิ้นสุด แต่ถ้าหากมีการกำหนดให้ 0!=1 ก็ทำให้การเรียกตัวเองสิ้นสุดลงเมื่อมีการเรียกลำดับที่ 0

ตัวอย่างฟังก์ชันหาค่าแฟคตอเรียลของเลขจำนวนหนึ่ง (n) สามารถเขียนได้ดังนี้

```
ex    function fac(n:integer):real;
        begin
          if n=0 then
             fac:=1                (* สิ้นสุดการเรียกตัวเอง *)
          else
             fac:=n*fac(n-1)       (* การเรียกตัวเอง *)
        end;
```

ยกตัวอย่างเช่น ในกรณีที่ต้องการคำนวณค่า 2! เมื่อมีการเรียกฟังก์ชัน fac(2) การดำเนินการในฟังก์ชัน fac จะมีลักษณะเป็นดังที่เห็น



จากรูปที่เห็นข้างบน เมื่อมีการเรียก fac(2) ฟังก์ชัน fac(n) ก็จะดำเนินการเรียกตัวเอง โดยที่ค่าพารามิเตอร์ภายในวงเล็บจะลดลงจนกระทั่งเป็น 0 กล่าวคือ ฟังก์ชัน fac เมื่อ n เท่ากับ 2 จะเรียกตัวเองที่ค่า n=1 (fac(1)) fac(1) ก็จะเรียก fac(0) ซึ่งได้มีการให้ค่าเป็น 1 และก็เป็นการสิ้นสุดการเรียกตัวเองด้วย

เมื่อได้ค่า fac(0) แล้ว ขั้นตอนต่อไปก็คือ ส่งค่า fac(0) (มีค่าเท่ากับ 1) ไปที่ fac(1) เพื่อดำเนินการคำนวณค่า fac(1) ซึ่งก็คือ 1*fac(0) (มีค่าเท่ากับ 1) หลังจากนั้นก็ส่งค่า fac(1) ไปที่ fac(2) เพื่อทำการคำนวณ ซึ่งก็คือ 2*1 ขั้นตอนสุดท้ายคือ ส่งค่าที่คำนวณได้ไปที่ฝ่ายเรียก

หมายเหตุ (1) - (6) ในรูปข้างบนคือขั้นตอนการดำเนินการของฟังก์ชัน fac

ตัวอย่างที่ 10.1 การคำนวณค่าแฟคตอเรียลตั้งแต่ 0! ถึง 20!

```
program factorial;
var n:integer;

   function fac(n:integer):real;
    begin
      if n=0 then
         fac:=1                        ............... 1
      else
         fac:=n*fac(n-1)               ............... 2
    end;
begin
   for n:=0 to 20 do
      writeln(n:3,'!=',fac(n):20:0) ........... 3
end.
```

1 กำหนดค่า 0! ให้เท่ากับ 1

2 สร้างโพรซิเจอร์เรียกตัวเอง

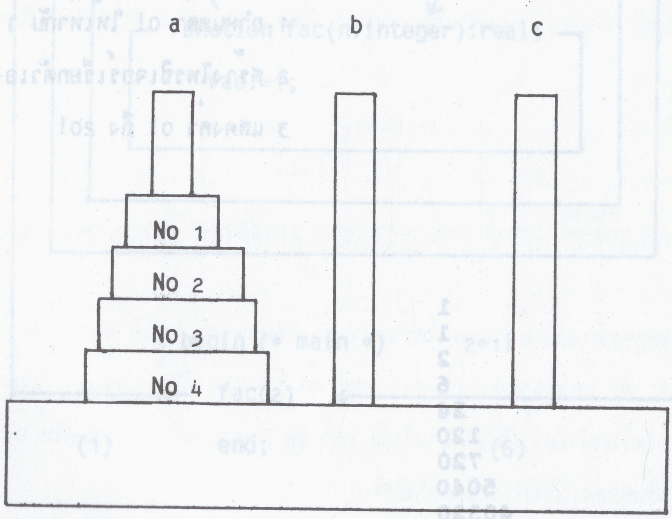
3 แสดงค่า 0! ถึง 20!

```
  0!=                   1
  1!=                   1
  2!=                   2
  3!=                   6
  4!=                  24
  5!=                 120
  6!=                 720
  7!=                5040
  8!=               40320
  9!=              362880
 10!=             3628800
 11!=            39916800
 12!=           479001600
 13!=          6227020800
 14!=         87178291200
 15!=       1307674368000
 16!=      20922789888000
 17!=     355687428100000
 18!=    6402373705700000
 19!=  121645100410000000
 20!= 2432902008200000000
```

< หอฮานอย >

นอกจากการหาค่าแฟคตอเรียลแล้ว การหาคำตอบจากปัญหา หอฮานอย ก็สามารถใช้โพรซิเจอร์เรียกตัวเองมาใช้ได้ รายละเอียดของปัญหามีดังนี้คือ

- มีเสาอยู่ 3 เสา คือ เสา a เสา b และเสา c
- ที่เสา a มีจานซึ่งมีรูตรงกลางเรียงกันอยู่ โดยจานแต่ละใบมีขนาดไม่เท่ากัน วิธีการเรียงของจานที่เสา a นั้น เรียงโดยจากใบเล็กไปหาใบใหญ่
- ให้ย้ายจานจากเสา a ไปที่เสา b โดยมีข้อแม้ว่า ย้ายจานได้ที่ละใบ ระหว่างที่ย้ายห้ามเอาจานใบใหญ่วางบนจานใบเล็ก
- การย้ายสามารถใช้เสา c ช่วยได้



< วิธีคิด >

ในกรณีที่มีจานเพียงใบเดียวที่เสา a

ย้ายใบที่ 1 จากเสา a ไปที่เสา b (ก)

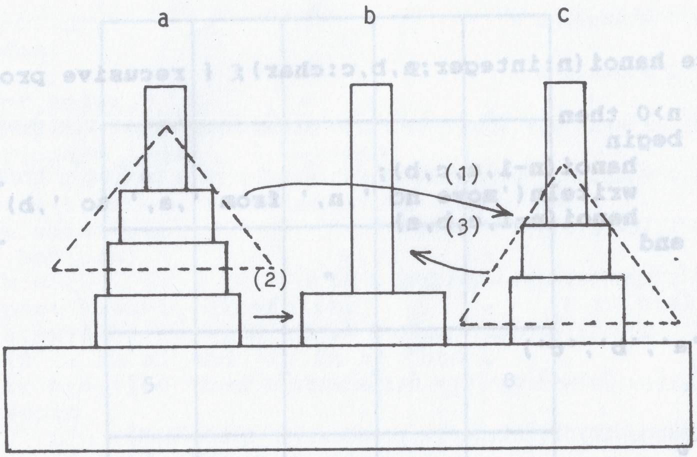
ในกรณีที่มีจาน 2 ใบที่เสา a

ย้ายใบที่ 1 จากเสา a ไปที่เสา c

ย้ายใบที่ 2 จากเสา a ไปที่เสา b (ข)

ย้ายใบที่ 1 จากเสา c ไปที่เสา b

ต่อไปลองดูว่าถ้ามีจาน 3 ใบที่เสา a จะทำอย่างไร



(1) ย้าย 2 ใบแรกจากเสา a ไปที่เสา c โดยใช้วิธี (ข)

(2) ย้ายใบที่ 3 ไปไว้ที่เสา b

(3) ย้าย 2 ใบแรกจากเสา c ไปที่เสา b โดยใช้วิธี (ข) เช่นเดียวกับข้อ (1)

จะเห็นได้ว่าขั้นตอนที่ (1) และ (3) ใช้วิธี (ข) หรือการเรียกตัวเองมาใช้โดยมีขั้นตอนดังนี้คือ ย้ายจาน n-1 ใบจากเสา a ไปที่เสา c หลังจากนั้นย้ายใบล่างสุดจากเสา a ไปที่เสา b ขั้นตอนสุดท้าย คือ ย้ายจาน n-1 ใบจากเสา c ไปที่เสา b

ถ้าเขียนเป็นภาษา Pascal ก็จะมีลักษณะเป็น

hanoi(n,a,b,c); (* โพรซิเจอร์ ย้ายจาน n ใบจากเสา a ไปเสา b *)

hanoi(n-1,a,c,b); (* โพรซิเจอร์ ย้ายจาน n-1 ใบจากเสา a ไปเสา c *)

hanoi(n-1,c,b,a); (* โพรซิเจอร์ ย้ายจาน n-1 ใบจากเสา c ไปเสา b *)

โพรซิเจอร์ hanoi(n,a,b,c) มี 4 พารามิเตอร์ n คือจำนวนจาน a,b,c คือเสาทั้ง 3 โดยมีกำหนดให้จานทั้งหมดวางเรียงกันอยู่บนเสา a และเตรียมที่จะย้ายไปเสา b

hanoi(n-1,a,c,b) เป็นการเรียกตัวเองในโพรซิเจอร์ hanoi เพื่อย้ายจาน n-1 ใบจากเสา a ไปเสา c ส่วน hanoi(n-1,c,b,a) เป็นการเรียกตัวเองเพื่อย้ายจาน n-1 ใบจากเสา c ไปเสา b

ตัวอย่างที่ 10.2 การแก้ปัญหาหอฮานอยโดยวิธีการเรียกตัวเอง

```
program hanoigame;
const n=4;

  procedure hanoi(n:integer;a,b,c:char); { recusive procedure }
    begin
      if n>0 then
        begin
          hanoi(n-1,a,c,b);                                        ....... 1
          writeln('move no ',n,' from ',a,' to ',b);               ....... 2
          hanoi(n-1,c,b,a)                                         ....... 3
        end
    end;

begin
  hanoi(n,'a','b','c')
end.
```

1 ย้ายจาน n-1 ใบจากเสา a ไปเสา c

2 ย้ายจานใบล่างสุดจากเสา a ไปเสา b

3 ย้ายจาน n-1 ใบจากเสา c ไปเสา b

```
move no 1 from a to c
move no 2 from a to b
move no 1 from c to b
move no 3 from a to c
move no 1 from b to a
move no 2 from b to c
move no 1 from a to c
move no 4 from a to b
move no 1 from c to b
move no 2 from c to a
move no 1 from b to a
move no 3 from c to b
move no 1 from a to c
move no 2 from a to b
move no 1 from c to b
```

< ม้าเดินทาง >

ม้าในหมากรุกมีตาเดินดังที่เห็นจากตารางข้างล่าง

	3		2	
4				1
		*		
5				8
	6		7	

ปัญหานี้คือให้เดินม้าจากตำแหน่งหนึ่งให้ครบทุกตำแหน่งโดยห้ามซ้ำตาเก่า การแก้ปัญหานี้สามารถใช้การเรียกตัวเองมาใช้ เพราะต้องเปลี่ยนตำแหน่งปัจจุบันและย้อนกลับไปตาเก่า

< อัลกอริทึม >

```
procedure try;                                  (* ชื่อโพรซิเจอร์        *)
begin initialize selection of moves;            (* กำหนดตำแหน่งแรก       *)
   repeat select next position;                 (* เลือกตาเดิน           *)
      if acceptable then                        (* ถ้ามีตาเดิน           *)
         begin record move;                     (* บันทึกตำแหน่ง          *)
            if board not full then              (* ถ้ายังไม่หมดกระดาน    *)
               begin try next move;             (* เดินต่อ               *)
                  if not successful then        (* ถ้าไม่มีตาเดิน         *)
                     erase previous record      (* ลบตาเดินเก่าทิ้ง       *)
               end
         end;
   until successful or no more candidates       (* จนกว่าจะสำเร็จ         *)
end;
```

ถ้ากำหนดให้

k : จำนวนครั้งที่เดิน

u : ตำแหน่งแกน x

v : ตำแหน่งแกน y

h[u,v] : ตำแหน่ง u,v เป็นตาเดินที่เท่าใด ถ้าเป็น 0 แสดงว่ายังไม่ได้เดินผ่าน

q1 : ตัวแปรแบบบูลีนเป็นค่าแสดงว่ามีตาเดินต่อไปหรือไม่ ถ้ามีให้เป็น true

s : เซท [1..5]

a[1..8] : แบบการเดินของม้าทางแกน x ทั้ง 8

b[1..8] : แบบการเดินของม้าทางแกน y ทั้ง 8

เมื่อเขียนเป็นโพรซิเจอร์ try ก็จะมีลักษณะเป็น

```
procedure try(i:integer;x,y:index;var q:boolean);
var k,u,v:integer;
    q1:boolean;
begin
    K;=0;
    repeat
        k:=k+1; q1:=false;
        u:=x+a[k]; v:=y+b[k];
        if (u in s) and (v in s) then
        if h[u,v]=0 then
            begin h[u,v]:=i;
                if i<5*5 then
                    begin try(i+1,u,v,q1);
                        if not q1 then h[u,v]:=0
                    end else q1:=true
            end
    until q1 or (k=8);
    q:=q1
end;
```

ตัวอย่างที่ 10.3 การแก้ปัญหาม้าเดินทางโดยใช้โพรซิเจอร์เรียกตัวเอง

```
program knight;
const n=5;
type index=1..n;
var i,j:index;
    q:boolean;
    s:set of index;
    a,b:array[1..8] of integer;
    h:array[index,index] of integer;

procedure try(i:integer;x,y:index;var q:boolean);
    var k,u,v:integer;
        q1:boolean;
    begin k:=0;
        repeat k:=k+1; q1:=false;                            { initialize }
          u:=x+a[k]; v:=y+b[k];
          if (u in s) and  (v in s) then
          if h[u,v]=0 then
          begin
             h[u,v]:=i;                                      { define new position }
             if i<25 then
                  begin   try(i+1,u,v,q1);                   { recursive procedure }
                     if not q1 then
                        h[u,v]:=0;                           { erase the previous }
                  end
                else q1:=true;
          end
        until q1 or (k=8);
        q:=q1
    end;
begin
    s:=[1,2,3,4,5];                                          ........... 1
    a[1]:= 2; b[1]:= 1;   a[2]:= 1; b[2]:= 2;
    a[3]:=-1; b[3]:= 2;   a[4]:=-2; b[4]:= 1;                ........... 2
    a[5]:=-2; b[5]:=-1;   a[6]:=-1; b[6]:=-2;
    a[7]:= 1; b[7]:=-2;   a[8]:= 2; b[8]:=-1;
  for i:=1 to n do
     for j:=1 to n do h[i,j]:=0;                             ........... 3
  h[1,1]:=1; try(2,1,1,q);                                   ........... 4
  if q then
     for i:=1 to n do
         begin
           for j:=1 to n do
           write(h[i,j]:5);
           writeln
         end
     else write('no solution!')
end.
```

1	6	15	10	21
14	9	20	5	16
19	2	7	22	11
8	13	24	17	4
25	18	3	12	23

1 กำหนดขอบเขตตารางหมากรุก

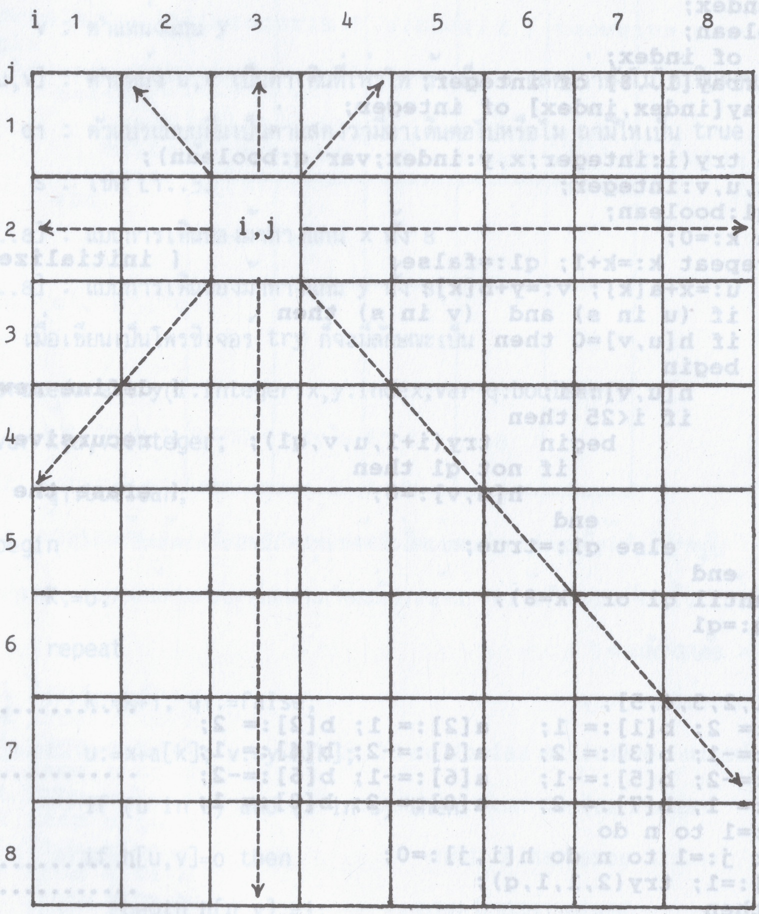
2 กำหนดแบบการเดินม้าทั้ง 8

3 ให้ h[i,j] ทุกค่าเท่ากับ 0

4 เรียกโพรซิเจอร์

< แปดควีน >

ควีน (queen) ในหมากรุกสากลสามารถเดินได้แปดทิศ เป็นระยะทางยาวเท่าใดก็ได้ ปัญหาคือ ให้วางควีน 8 ตัวบนกระดาน 8*8 โดยไม่ให้ควีนตัวใดตัวหนึ่งอยู่บนตาเดินของควีนตัวอื่นๆ



จากรูปถ้ามีควีนอยู่ที่ตำแหน่ง (i,j) เงื่อนไขในการวางควีนตัวต่อไปก็คือห้ามวางตามแนวตั้งเท่ากับ i และแนวนอนเท่ากับ j นอกจากนี้ยังมีเงื่อนไขทางด้านแทยงอีกด้วย จากรูปจะเห็นได้ว่า ควีนอยู่บนตำแหน่ง (3,2) การตรวจเงื่อนไขแนวแทยงซ้าย (⤡) ตรวจโดยบวกตำแหน่ง (i,j) ทั้ง 2 ด้วยกัน ถ้าตำแหน่งที่ต้องการวางควีนตัวใหม่บวกได้ค่าเท่ากับตัวที่วางอยู่เดิมแสดงว่าวางไม่ได้ จากรูปผลบวกของตำแหน่งทั้งสองคือ 3+2=5 ดังนั้น ถ้าเลือกตำแหน่ง (2,3),(4,1).. ควีนใหม่ก็จะอยู่บนตาเดินของควีนเก่า ต่อไปการตรวจเงื่อนไขแนวแทยงขวา (⤢) ก็ใช้แกนนอนลบแกนตั้ง (3-2=1) แสดงว่า ไม่สามารถวางควีนที่ตำแหน่ง (2,1),(4,3) ด้วยเหตุผลเดียวกัน

< อัลกอริทึม >

```
procedure try(i:integer);                    (* ชื่อโพรซีเจอร์        *)
begin
  initialize positions;                      (* กำหนดตำแหน่งแรก     *)
  repeat make next selection;                (* เลือกตาวาง            *)
    if safe then                             (* ถ้ามีที่ว่าง            *)
      begin
        setqueen;                            (* บันทึกตำแหน่ง          *)
        if i<8 then                          (* ถ้ายังไม่ครบแปดตัว    *)
          begin
            try(i+1);                        (* วางตัวต่อไป           *)
            if not successful then           (* ถ้าวางไม่ได้           *)
              remove queen                   (* ลบควีนทิ้ง            *)
          end
      end;
  until successful or no more positions      (* จนกว่าสำเร็จหรือครบ *)
end;
```

ถ้ากำหนดให้

```
x:array[1..8] of integer;
a:array[1..8] of boolean;
b:array[b1..b2] of boolean;
c:array[c1..c2] of boolean;
```

โดยให้

x[i] คือ ตำแหน่งแนวนอนของควีนแนวตั้งที่ i

a[j] คือ ค่าบูลีนแสดงแนวตั้งที่ j (* เป็น true ถ้าไม่มีควีนอยู่ *)

b[k] คือ ค่าบูลีนแสดงแนวแทยงซ้ายที่ k (* เป็น true ถ้าไม่มีควีนอยู่ *)

c[k] คือ ค่าบูลีนแสดงแนวแทยงขวาที่ k (* เป็น true ถ้าไม่มีควีนอยู่ *)

จากเงื่อนไขที่ว่า

ผลบวกของตำแหน่งแนวนอนกับแนวตั้งเท่ากันจะอยู่บนเส้นแทยงซ้าย (/) เดียวกัน

ผลต่างของตำแหน่งแนวนอนกับแนวตั้งเท่ากันจะอยู่บนเส้นแทยงขวา (\) เดียวกัน

ถ้ากำหนดให้ x[i]:=j; ค่า a,b,c จะมีค่าเป็น

x[i]:=j; a[j]:=false; b[i+j]:=false; c[i-j]:=false;

ยกตัวอย่างเช่นถ้าวางตัวแรกที่แนว (นอน) ที่ 2 (x[1]:=2;) ผลจะได้เป็น

a[2]:=false; (ห้ามวางควีนที่แนวนอนที่ 2)

b[i+j]:=false; (ห้ามวางควีนตัวที่ 2,3,4,..8 ที่แนวนอนที่ 3,4....9)

c[i-j]:=false; (ห้ามวางควีนตัวที่ 2,3,4,..8 ที่แนวนอนที่ 1,0...-6)

เมื่อต้องการตรวจว่า สามารถวางควีน x[i] ที่ j ได้หรือไม่เพียงใช้เงื่อนไข

a[j] and b[i+j] and c[i-j]

หมายเหตุ ตำแหน่งมากกว่า 8 หรือติดลบ สาเหตุจากการคำนวณจึงออกนอกกระดาน

ถ้าเขียนเป็นโพรซิเจอร์จะเขียนได้เป็น

```
procedure try(i:integer);
var j,k:integer;
begin
    for j:=1 to 8 do
      if a[j] and b[i+j] and c[i-j] then
      begin x[i]:=j;a[j]:=false; b[i+j]:=false; c[i-j]:=false;
          if i<8 then try(i+1)
              else begin
                      for k:=1 to 8 do
                          write(x[k]:4);
                      writeln
                   end;
          a[j]:=true; b[i+j]:=true; c[i-j]:=true
      end
end;
```

ตัวอย่างที่ 10.4 การหาตำแหน่งแปดควีน

```
program eightqueen(output);
 type n=1..8;
 var i:integer;
    q:boolean;
    x:array[n] of n;
    a:array[n] of boolean;
    b:array[2..16] of boolean;
    c:array[-7..7] of boolean;
   procedure try(i:integer);
     var j,k:integer;
     begin
       for j:=1 to 8 do
          if a[j] and b[i+j] and c[i-j] then
          begin
            x[i]:=j;
            a[j]:=false;
            b[i+j]:=false;
            c[i-j]:=false;
            if i<8 then
               try(i+1)
            else
               begin
                 for k:=1 to 8 do
                   write(x[k]:4);
                   writeln
               end;
            a[j]:=true; b[i+j]:=true; c[i-j]:=true
          end
   end;
begin (* main *)
   for i:=1 to 8 do a[i]:=true;
   for i:=2 to 16 do b[i]:=true;
   for i:=-7 to 7 do c[i]:=true;
   try(1);
end.
```

```
1   5   8   6   3   7   2   4
1   6   8   3   7   4   2   5
1   7   4   6   8   2   5   3
1   7   5   8   2   4   6   3
2   4   6   8   3   1   7   5
2   5   7   1   3   8   6   4
2   5   7   4   1   8   6   3
2   6   1   7   4   8   3   5
2   6   8   3   1   4   7   5
2   7   3   6   8   5   1   4
2   7   5   8   1   4   6   3
2   8   6   1   3   5   7   4
```

หมายเหตุ คำตอบข้างบนเป็นเพียงบางส่วนเท่านั้น

10.2 โครงสร้างข้อมูล (data structure)

โครงสร้างของข้อมูลมีความสำคัญต่อโปรแกรมและอัลกอริทึมเป็นอย่างมากอาจกล่าวได้ว่า

[โครงสร้างข้อมูล + อัลกอริทึม = โปรแกรม]

จะเห็นได้ว่า โครงสร้างแบบข้อมูลมีส่วนสัมพันธ์กับอัลกอริทึมเป็นอย่างมาก นั่นก็หมายถึง ถ้าเลือกโครงสร้างข้อมูลที่เหมาะสม ก็จะได้โปรแกรมที่ดีด้วย โครงสร้างข้อมูลแบ่งออกได้ดังนี้คือ

1.แบบตาราง table ลักษณะของข้อมูลจะเป็น

1	3	6	4	7	5	3
2	3	9	0	9	2	1

2.แบบแสตก (stack) หรือแบบ LIFO (last in first out)

อินพุท

3	4	5	6

เอาท์พุท

6	5	4	3

3. แบบคิว (queue) หรือแบบ FIFO (first in first out)

อินพุท

3	4	5	6

เอาท์พุท

3	4	5	6

4.แบบลิสท์ (list)

5.แบบทรี (tree)

6.แบบกราฟ (graph)

โครงสร้างข้อมูลเหล่านี้ ต้องนำข้อมูลแบบต่างๆ ได้แก่ อาร์เรย์ เรคอร์ด พอยน์เตอร์ มาประยุกต์ใช้ ภาษา Pascal มีแบบข้อมูลมาก จึงเหมาะแก่การประยุกต์งานดังกล่าว

ในบทนี้จะกล่าวถึง ข้อมูลแบบลิสท์ (tree) และทรี (tree) ซึ่งเป็นโครงสร้างข้อมูลที่นำเอาข้อมูลแบบเรคอร์ดและแบบพอยน์เตอร์มาประยุกต์

10.3 ข้อมูลแบบพอยน์เตอร์

ข้อมูลแบบอาร์เรย์เหมาะกับการใช้ตัวแปรคราวละมากๆ แต่ต้องกำหนดขอบเขตของจำนวนตัวแปรทำให้ไม่สามารถใช้ตัวแปรเกินที่ขอบเขตที่กำหนดได้ แต่ข้อมูลแบบพอยน์เตอร์สามารถใช้ตัวแปรได้มากน้อยตามแต่ผู้ใช้ต้องการ จึงเหมาะกับข้อมูลที่มีจำนวนไม่แน่นอน

< การกำหนดแบบข้อมูลและตัวแปรพอยน์เตอร์ >

ลักษณะพิเศษของข้อมูลพอยน์เตอร์ คือ นอกจากสามารถให้หรืออ้างค่าต่างๆแล้ว ยังสามารถกำหนดแบบข้อมูลโดยใช้สัญลักษณ์ ^ ดังที่เห็น

type แบบข้อมูล = ^แบบข้อมูลที่ต้องการชี้

แบบข้อมูลที่ต้องการชี้ต้องเป็นแบบข้อมูลประเภทพื้นฐาน ซึ่งเหมือนกับแบบพารามิเตอร์ กล่าวคือต้องกำหนดแบบข้อมูลขึ้นมาเอง ในกรณีที่ใช้ข้อมูลที่นอกเหนือประเภทพื้นฐาน

หลังจากกำหนดแบบข้อมูลแล้ว ก็สามารถประกาศตัวแปรพอยน์เตอร์ได้ดังนี้

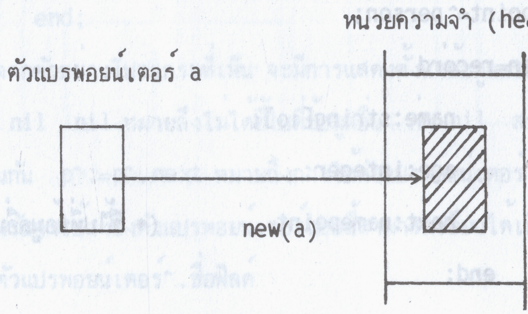
var ตัวแปร:แบบข้อมูลพอยน์เตอร์;

```
ex    type point=^name;
           name=string[10];
      var a:point;
```

เมื่อกำหนดแบบข้อมูลและตัวแปรเรียบร้อยแล้ว point จะเป็นแบบข้อมูลพอยน์เตอร์ที่สามารถเก็บอักขระได้ 10 ตัวอักขระ และ a ก็จะเป็นตัวแปรแบบ point

< การจองหน่วยความจำโดยคำสั่ง new >

การประกาศตัวแปรเพียงอย่างเดียวไม่พอต่อการให้หรืออ้างค่าได้ เพียงเป็นการบอกถึงแบบข้อมูลที่ต้องการใช้เท่านั้น ในกรณีที่ต้องการใช้ตัวแปร a ต้องกำหนดตำแหน่งหน่วยความจำโดยคำสั่ง new ซึ่งเป็นโพรซิเจอร์มาตรฐาน



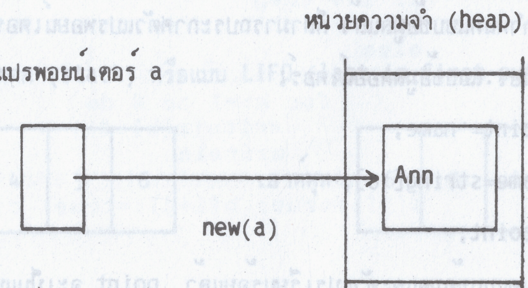
คำสั่ง new เป็นคำสั่งในการกำหนดตำแหน่งหน่วยความจำส่วนที่เรียกว่า ฮีพ (heap) และส่งตำแหน่งดังกล่าวไปที่ a เพื่อใช้ในการเก็บค่า เมื่อใช้คำสั่ง new ขึ้นใหม่ก็จะมีการกำหนดตำแหน่งหน่วยความจำขึ้นมาใหม่ทุกครั้ง คำสั่ง new นี้ไม่ใช่เป็นการให้ค่ากับตัวแปร แต่ดำเนินการกำหนดตำแหน่งหน่วยความจำซึ่งเรียกว่า ฮีพ (heap) ให้กับตัวแปรพอยน์เตอร์ a เท่านั้น

< การอ้างและให้ค่ากับตัวแปรพอยน์เตอร์ >

เมื่อกำหนดตัวแปรพอยน์เตอร์และใช้คำสั่ง new(a) เพื่อกำหนดตำแหน่งหน่วยความจำให้กับ a ก็สามารถใช้ตัวแปร a ได้โดยใช้สัญลักษณ์ ^ ดังข้างล่าง

```
a^:='Ann';
```

ภายในหน่วยความจำจะมีข้อมูล 'Ann' อยู่



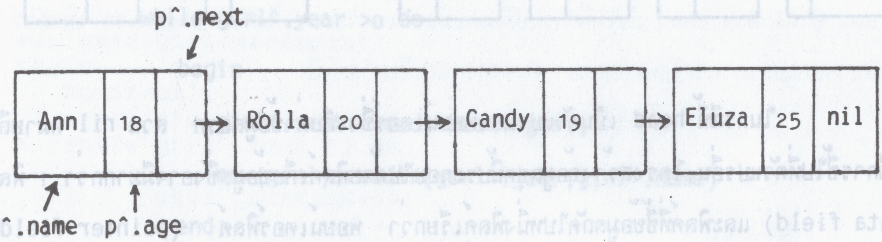
< แบบข้อมูลพอยน์เตอร์ที่สามารถชี้ข้อมูลอื่น >

ตัวอย่างข้างบนเป็นการใช้ตัวแปรแบบพอยน์เตอร์เพียงหนึ่งตัวแปร ถ้าหากมีการนำเอาข้อมูลแบบเรคอร์ดมามาประยุกต์ใช้ก็จะสามารถทำให้ตัวแปรปัจจุบันชี้ไปที่ตัวแปรที่อยู่ถัดไปได้ ซึ่งเป็นโครงสร้างข้อมูลที่มีประโยชน์สำหรับการเก็บข้อมูลพอยน์เตอร์ที่มีมากกว่า 1 ข้อมูล โครงสร้างข้อมูลแบบนี้สามารถเขียนได้ดังนี้คือ

ex
```
type namepoint=^person;
     person=record
            name:string[10];
            age:integer;
            next:namepoint          (* ชี้ไปที่ข้อมูลอื่น *)
     end;
```

ข้อมูลพอยน์เตอร์แบบนี้ต่างไปจากข้อมูลพอยน์เตอร์ที่แล้วมา ซึ่งจะเห็นว่ามีฟิลด์ next เพิ่มขึ้นมาอีกฟิลด์หนึ่งเป็นแบบ namepoint (เป็นแบบพอยน์เตอร์) ฟิลด์ next นี้จะทำหน้าที่ชี้ข้อมูลแบบ person อื่น ลักษณะของข้อมูลเรคอร์ดที่มีฟิลด์ที่ชี้ข้อมูลอื่นเป็นลักษณะพิเศษของข้อมูลพอยน์เตอร์ ซึ่งเป็นประโยชน์ต่อการสร้างโครงสร้างข้อมูลลิสท์ (list) และ ทรี (tree)

จากโครงสร้างข้อมูลดังกล่าว ทำให้สามารถสร้างแบบข้อมูลดังที่เห็นข้างล่าง



ตัวแปรพอยน์เตอร์ p

จะเห็นได้ว่า ฟิลด์ next เป็นฟิลด์ที่มีไว้สำหรับชี้ข้อมูลถัดไป ลักษณะของข้อมูลชุดนี้จะดูราวกับว่ามีข้อมูลของชื่อและอายุเรียงยาวจนถึง Eluza การกำหนดตำแหน่งหน่วยความจำใหม่นั้นใช้คำสั่ง new เมื่อใส่ข้อมูลเรียบร้อยแล้ว วิธีการแสดงออกก็สามารถเขียนได้ดังนี้คือ

ถ้าประกาศให้ p เป็นข้อมูลแบบ namepoint โดยใช้คำสั่ง

```
var p:namepoint;
```

การแสดงเอาท์พุทก็จะแสดงได้เป็น

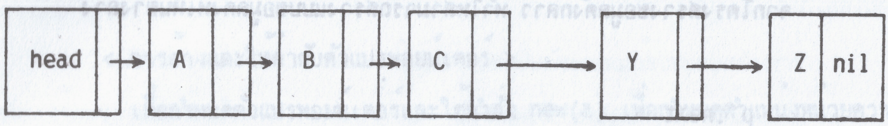
```
ex  p:= ข้อมูลตัวแรก;
    while p<>nil do                    (* ดำเนินการไปจนถึงตัวสุดท้าย *)
      begin
        writeln(p^.name,p^.year);  (* แสดงฟิลด์ชื่อและอายุ        *)
        p:=p^.next;                   (* ให้ p เท่ากับข้อมูลที่ p ชี้อยู่ *)
      end;
```

จากตัวอย่างโปรแกรมที่เห็น จะมีการแสดงตั้งแต่ข้อมูลแรกจนถึงข้อมูลสุดท้าย ซึ่งมีฟิลด์ next เท่ากับ nil nil หมายถึงไม่ได้ชี้ไปที่ข้อมูลอื่น ค่า nil สามารถใช้ในการเปรียบเทียบหรืออ้างได้เช่นกัน p^:=p^.next หมายถึงการให้ตัวแปรพอยน์เตอร์ p ในปัจจุบันเท่ากับตัวแปรที่ p ชี้อยู่ สำหรับการอ้างถึงตัวแปรพอยน์เตอร์แบบนี้สามารถเขียนได้เป็นลักษณะที่เห็น

ตัวแปรพอยน์เตอร์^.ชื่อฟิลด์

10.4 ข้อมูลแบบลิสท์ (list)

ข้อมูลพอยน์เตอร์ที่เป็นที่มีฟิลด์ชี้ไปที่ข้อมูลอื่นเรียกว่า ข้อมูลแบบลิสท์ (list) และมีลักษณะดังที่เห็น

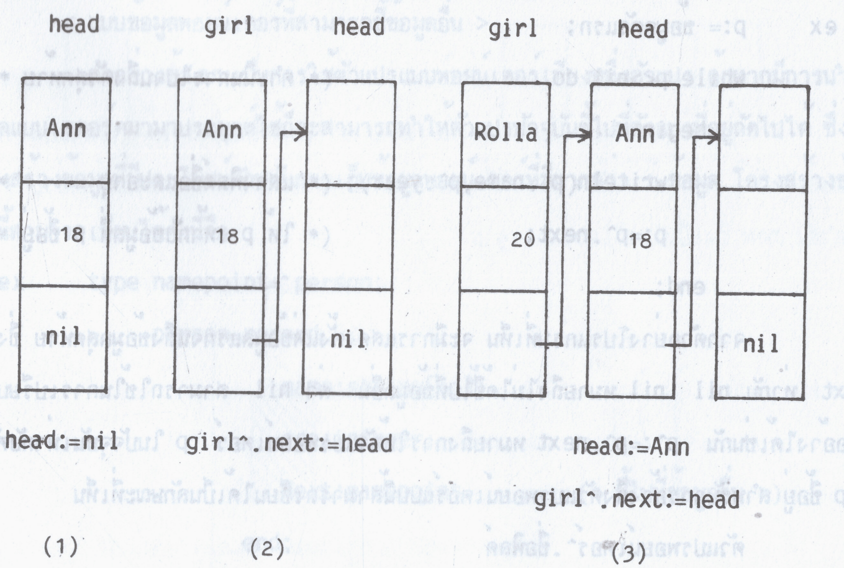


ในกรณีนี้ head เป็นตัวแปรแบบพอยน์เตอร์ซึ่งเก็บค่าข้อมูลแรก ส่วน nil หมายถึงไม่มีการชี้ไปที่ตัวแปรอื่น โครงสร้างข้อมูลชุดนี้ประกอบไปด้วยฟิลด์เก็บข้อมูลซึ่งอาจมีมากกว่า 1 ฟิลด์ (data field) และฟิลด์ที่ชี้ข้อมูลถัดไปหนึ่งฟิลด์เรียกว่า พอยน์เตอร์ฟิลด์ (pointer field) โครงสร้างข้อมูลแบบนี้สามารถตัดหรือแทรกข้อมูลได้ โดยเปลี่ยนเพียงพอยน์เตอร์ฟิลด์เท่านั้น

< โครงสร้างข้อมูลแบบ LIFO >

สมมุติว่าต้องการสร้างข้อมูลที่มีฟิลด์ชื่อและอายุ โดยใส่อินพุทให้กับตัวแปร girl และ head โดยให้เอาท์พุทมีลักษณะ LIFO (last in first out) ดังตัวอย่าง

อินพุท Ann,18 Rolla,20 เอาท์พุท Rolla,20 Ann,18



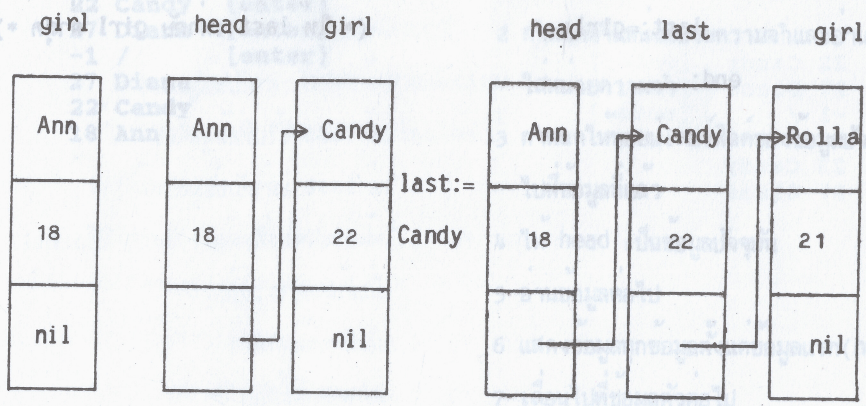
ก่อนอื่นต้องประกาศตัวแปรที่เก็บข้อมูลชุดนี้ (girl,head) และให้ head เป็น nil หลังจากใส่ข้อมูลแรก (Ann) แล้ว ให้ Ann ชี้มาที่ head หลังจากนั้นให้ head เท่ากับข้อมูลล่าสุด (Ann) หมายเลข (1),(2),(3) หมายถึงขั้นตอนการเปลี่ยนแปลงของตัวแปร girl และ head เมื่อเขียนเป็นภาษา Pascal จะเขียนได้ดังนี้

```
head:=nil;
new(girl);readln(girl^.year,girl^.name);
while girl^.year >o do
  begin
    girl^.next:=head; head:=girl;
    new(girl);readln(girl^.year,girl^.name)
    end;
end;
```

< โครงสร้างข้อมูลแบบ FIFO >

ลักษณะของโครงสร้างข้อมูลแบบนี้คือ อินพุทและเอาท์พุทเป็นแบบ FIFO (first in first out) ดังตัวอย่าง

อินพุท Ann,18 Candy,22 Rolla,21 เอาท์พุท Ann,18 Candy,22 Rolla,21



head:=girl last:=girl (1)

last^.next:=girl (2)

last^.next:=girl (3)

ก่อนอื่นต้องประกาศตัวแปรพอยน์เตอร์ 3 ตัวแปรคือ (girl last head) ให้ head เป็นข้อมูลแรก last เป็นข้อมูลตัวสุดท้ายและ girl คือข้อมูลที่ใส่เข้ามาใหม่ เมื่อให้ค่าแก่ girl last จะชี้ไปที่ข้อมูลล่าสุด (girl) ต่อจากนั้นให้ last เท่ากับข้อมูลล่าสุดนั้นด้วย ดังนั้นเมื่อมีการใส่ข้อมูลใหม่เข้ามาที่ girl last จะชี้ไปที่ girl หลังจากนั้น last จะเท่ากับ girl เป็นเช่นนี้ทุกครั้ง ซึ่งทำให้ข้อมูลที่ใส่เข้าไปมีลักษณะต่อเนื่องไป ตั้งแต่ข้อมูลแรกจนถึงข้อมูลสุดท้าย สำหรับการดำเนินการจะเป็นไปตามหมายเลข (1),(2),(3) ซึ่งทำให้เห็นได้ชัดถึงการเปลี่ยนแปลงของ last และ girl ส่วน head นั้นไม่มีการเปลี่ยนแปลงค่าเพราะต้องเก็บค่าของข้อมูลแรก เพื่อการแสดงผลจากข้อมูลแรก ถ้าเขียนเป็นภาษา Pascal จะเขียนได้ดังนี้

```
new(girl);
readln(girl^.year,girl^.name);  (* ให้ค่า girl *)
head:=girl;
last:=girl;                     (* ให้ head last เป็น girl *)
while girl^.year >o do
  begin
    new(girl);
    readln(girl^.year,girl^.name) (* ให้ค่า girl *)
      last^.next:=girl;         (* ให้ last ชี้ไปที่ girl ล่าสุด *)
      last:=girl;               (* ให้ last เท่ากับ girl ล่าสุด *)
  end;
```

ตัวอย่างที่ 10.5 การเขียนโครงสร้างข้อมูลแบบ LIFO โดยใส่ข้อมูลของอายุและชื่อและแสดงผลออกมาโดยลำดับจากข้อมูลหลังสุด

```
program list1;
type namepoint=^person;
   person=record
              name:string[20];
              year:integer;
              next:namepoint
           end;
var head,girl:namepoint;
begin
   head:=nil;                                      .............. 1
   new(girl);readln(girl^.year,girl^.name);   ......... 2
   while girl^.year >0 do
     begin
        girl^.next:=head;                          ......... 3
        head:=girl;                                          4
        new(girl);readln(girl^.year,girl^.name)  ... 5
     end;
   girl:=head;
   while girl <> nil do
     begin
        writeln(girl^.year,girl^.name);            ........ 6
        girl:=girl^.next                           ............. 7
     end
end.
```

```
18 Ann     [enter]
22 Candy   [enter]
27 Diana   [enter]
-1 /       [enter]
27 Diana
22 Candy
18 Ann
```

1 กำหนดให้ head ค่าแรกเป็น nil

2 กำหนดตำแหน่งหน่วยความจำและอ่านข้อมูลใส่หน่วยความจำ

3 กำหนดให้พอยน์เตอร์ฟิลด์ของข้อมูลปัจจุบันชี้ไปที่ข้อมูลที่แล้ว

4 ให้ head เป็นข้อมูลปัจจุบัน

5 อ่านข้อมูลต่อไป

6 แสดงข้อมูลทุกข้อมูลตั้งแต่ข้อมูลแรก(head)

7 เลื่อนไปที่ข้อมูลตัวต่อไป

ตัวอย่างที่ 10.6 การเขียนโครงสร้างข้อมูลแบบ FIFO โดยใส่ข้อมูลของอายุและชื่อและแสดงผลออกมาโดยลำดับจากข้อมูลแรกสุด

```
program list2;
type namepoint=^person;
    person=record
                    name:string[20];
                    year:integer;
                    next:namepoint
                end;
var head,girl,last:namepoint;
begin
  new(girl);readln(girl^.year,girl^.name);   ............ 1
  head:=girl; last:=girl;                    ............ 2
  while girl^.year >0 do
    begin
      new(girl);readln(girl^.year,girl^.name);
      last^.next:=girl;                      ............ 3
      last:=girl;                            ............ 4
      last^.next:=nil;                       ............ 5
    end;
  girl:=head;
  while  girl^.next <> nil do                ............ 6
    begin
      writeln(girl^.year,girl^.name);        ............ 7
      girl:=girl^.next;
    end
end.
```

```
18 Ann     [enter]
22 Candy   [enter]
27 Diana   [enter]
-1 /       [enter]
18 Ann
22 Candy
27 Diana
```

1 อ่านข้อมูล

2 ให้ head เท่ากับข้อมูลแรก

3 ให้ last ชี้ไปที่ข้อมูลใหม่

4 ให้ last เท่ากับข้อมูลใหม่

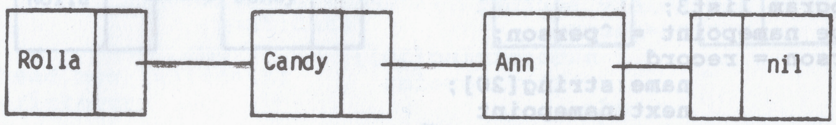
5 ให้พอยน์เตอร์ฟิลด์ของ last เป็น nil เมื่อสิ้นสุดการใส่ข้อมูล

6 กลับมาที่ข้อมูลแรก

7 แสดงผลทุกข้อมูล

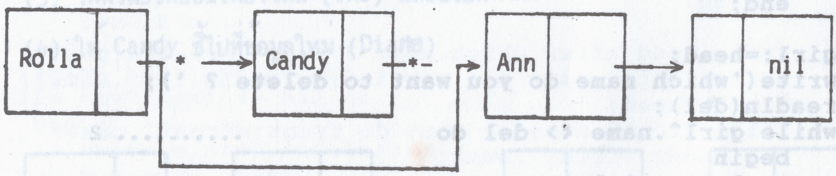
< การตัดข้อมูล >

ลักษณะพิเศษของโครงสร้างข้อมูลแบบลิสท์คือ การตัดและการแทรกข้อมูล การตัดข้อมูลใช้หลักการดังต่อไปนี้



ลักษณะข้อมูล

ถ้าต้องการตัดข้อมูล Candy ทิ้ง ก่อนอื่นต้องหาตำแหน่งของ Candy หลังจากนั้นจึงให้ข้อมูลก่อน Candy (ก็คือ Rolla) ชี้ไปที่ข้อมูลที่ Candy ชี้อยู่ (Ann)



(3)

ขั้นตอนการดำเนินการ

```
(1)     last:=Rolla;
(2)     girl:=Candy;
(3)     last^.next:=girl^.next;
```

ถ้าเขียนเป็นภาษา Pascal จะเขียนได้ดังนี้

```
girl:=head;
while girl<> 'Candy' do        (* วนลูปเพื่อหา Candy *)
begin
    last:=girl;                (* last = Rolla *)
    girl:=girl^.next           (* girl = Candy *)
end;
last^.next:=girl^.next;        (* last^.next = Ann *)
```

ตัวอย่างที่ 10.7 การตัดข้อมูลจากโครงสร้างข้อมูลแบบลิสท์

```
program list3;
 type namepoint = ^person;
 person = record
           name:string[20];
           next:namepoint
         end;
 var head,girl,last:namepoint;
   del:string[20];
 begin
   head:=nil;
   new(girl);readln(girl^.name);                    ......... 1
   while girl^.name <>'/' do
     begin
       girl^.next:=head;
       head:=girl;
       new(girl);readln(girl^.name);
     end;

   girl:=head;
   write('which name do you want to delete ? ');
   readln(del);
   while girl^.name <> del do                       ......... 2
     begin
       last:=girl;
       girl:=girl^.next;
     end;
  last^.next:=girl^.next;                           ......... 3

  girl:=head;
  while girl <> nil do
    begin
      writeln(girl^.name);                          ......... 4
      girl:=girl^.next;
    end
end.
```

```
Ann     [enter]
Candy   [enter]
Diana   [enter]
/  .    [enter]
which name to delete ? Candy
Diana
Ann
```

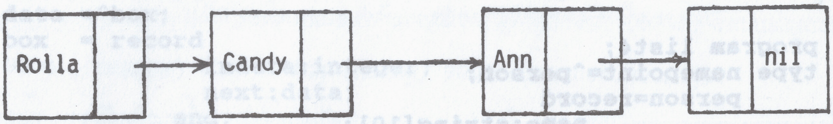
1 อ่านข้อมูลแบบ LIFO

2 วนหาชื่อที่ต้องการตัด เมื่อพบชื่อ Candy last จะเท่ากับชื่อก่อนหน้านี้ (Rolla) ส่วน girl จะเท่ากับ Candy

3 ให้ไฟล์พอยน์เตอร์ของ last ชี้ไปที่ girl^.next (Ann)

4 แสดงข้อมูลทั้งหมด (ไม่มี Candy อยู่)

< การแทรกข้อมูล >



ลักษณะข้อมูล

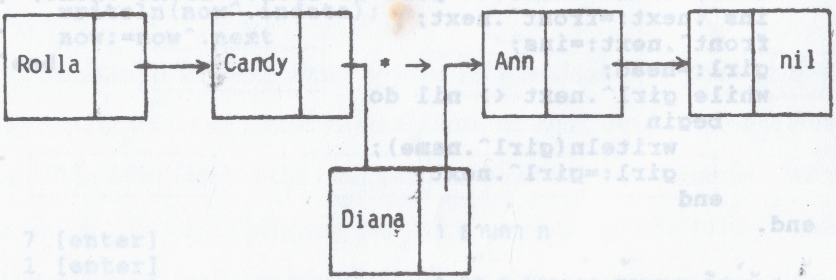
ถ้าต้องการแทรก Diana ระหว่าง Candy กับ Ann จะมีขั้นตอนดังนี้ คือ

(1) ดำเนินการหาตำแหน่งของ Candy

(2) ใช้คำสั่ง new เพื่อเก็บข้อมูลให้ตัวแปร ins (Diana)

(3) ให้ตัวแปรพอยน์เตอร์ใหม่ (ins) และชี้ไปที่ Ann

(4) ให้ Candy ชี้ไปที่ข้อมูลใหม่ (Diana)



(ins)

ถ้าเขียนเป็นภาษา Pascal จะเขียนได้ดังนี้

```
front:=head;
while girl <> 'Candy' do
    front:=front^.next;     (* หาตำแหน่งของ Candy *)
ins^.next:=front^.next;     (* ให้ ins ชี้ที่  Ann *)
front^.next:=ins;           (* ให้ Candy ชี้ที่ Diana *)
```

ตัวอย่างที่ 10.8 การแทรกข้อมูลเข้าในโครงสร้างข้อมูลแบบลิสท์

```
program list4;
type namepoint=^person;
     person=record
              name:string[10];
              next:namepoint
            end;
var head,girl,front,last,ins:namepoint;
    post,insert:string[10];
begin
   new(girl); readln(girl^.name);
   head:=girl; last:=girl;                                   ........... 1
   while girl^.name <> '/' do
     begin
        new(girl); readln(girl^.name);                      ........... 2
        last^.next:=girl;
        last:=girl;
        last^.next:=nil
     end;

   write('add after which name ? ');      readln(post);
   write('enter name '); new(ins) ;readln(ins^.name);
   front:=head;
   while front^.name <> post do front:=front^.next;   ... 3
   ins^.next:=front^.next;                                          ... 4
   front^.next:=ins;                                                ... 5
   girl:=head;
   while girl^.next <> nil do
     begin
        writeln(girl^.name);                                        ... 6
        girl:=girl^.next
     end
end.
```

```
Ann    [enter]
Candy  [enter]
Diana  [enter]
/      [enter]
add after which name ? Candy [enter]
enter name             Eluza [enter]
Ann
Candy
Eluza
Diana
```

1 ให้ head เป็นข้อมูลแรก

2 อ่านข้อมูลแบบ FIFO

3 วนหาชื่อที่ตำแหน่งที่ต้องการใส่ชื่อ เมื่อพบ front จะเท่ากับ Candy

4 ให้ ins ชี้ที่ Ann

5 ให้ front ชี้ไปที่ ins

6

ตัวอย่างที่ 10.9 การเรียงลำดับข้อมูลแบบลิสท์

```
program list5;
type data =^box;
    box  = record
                    indata:integer;
                    next:data;
                  end;
var head,now,last:data;
   n:integer;
begin
  new(now);
  now^.indata:=0; now^.next:=nil;
  readln(n);                                          ........................ 1
  while n <> 0 do
    begin
      last:=now;
      while last^.indata >n do last:=last^.next; .. 2
      new(head);
      head^:=last^;
      last^.indata:=n;                                .. 3
      last^.next:=head;                               .. 4
      readln(n);
    end;
  writeln('hear are output');
  while now^.next <> nil do
    begin
      writeln(now^.indata);
      now:=now^.next
    end
end.
```

```
7 [enter]
1 [enter]
2 [enter]
3 [enter]
5 [enter]
0 [enter]
hear are output
7
5
3
2
1
```

1 อ่านค่า n

2 ถ้า n มากกว่า ข้อมูลแบบลิสท์ตัวสุดท้าย ให้วนลูปเพื่อหาข้อมูลที่น้อยกว่า n ex ข้อมูลแบบลิสท์ 7 3 2 1 ถ้าอินพุทเป็น 5 last จะเท่ากับ 3

3 ทำการแทรกข้อมูล สมมุติข้อมูลชุดนี้เป็น 7 3 2 1 เมื่อใส่ข้อมูล 5 head จะเท่ากับ last (head จะเป็น 3) และ และ last จะเท่ากับ ข้อมูลใหม่ (5)

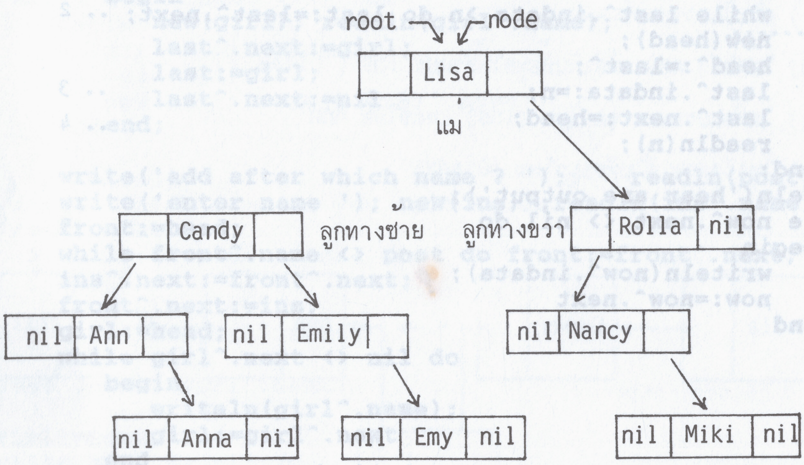
4 ให้ last ชี้ไปที่ head

10.5 ข้อมูลแบบทรี (tree)

ข้อมูลแบบลิสท์เป็นข้อมูลที่ต่อกันโดยมีลักษณะเป็นเส้นตรง ไม่มีทางแยกระหว่างข้อมูล แต่ข้อมูลแบบทรี (tree) เป็นข้อมูลที่มีโครงสร้างที่ทางแยกเหมือนกิ่งไม้

ข้อมูลแบบนี้ประกอบด้วย node (จุดแยก) และ branch (ทางแยก) node คือ จุดเชื่อมระหว่างข้อมูล ส่วน branch คือทางแยก ความสัมพันธ์ระหว่างข้อมูลมีหลักว่า ข้อมูลตอนต้น branch คือ แม่ (ancestor) ข้อมูลตอนปลายคือ ลูก (descendant)

เนื่องจากโครงสร้างข้อมูลแบบนี้ มีทางแยกมากกว่า 2 ทาง จึงสามารถกำหนดความสัมพันธ์ของแม่กับลูกได้ดังเช่น มากกว่าหรือน้อยกว่า เป็นต้น ในบทนี้จะกล่าวถึงการเรียงลำดับข้อมูลโดยวิธี binary tree



จากรูปข้างบนจะเห็นได้ว่า เป็นการเรียงลำดับข้อมูลโดยใช้ทางแยก 2 ทาง คือซ้ายและขวา ซึ่งมีความสัมพันธ์ดังนี้คือ

ลูกทางซ้าย <แม่ < ลูกทางขวา

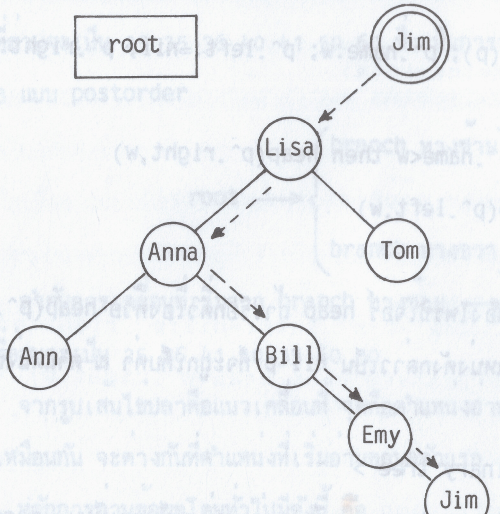
การกำหนดข้อมูลแบบ binary tree สามารถเขียนเป็นลักษณะที่เห็น

```
type point =^person;
        person=record
                left:point;            (* ฟิลด์ชี้ลูกทางซ้าย *)
                name:string[10];       (* ฟิลด์ข้อมูลชื่อ    *)
                right:point            (* ฟิลด์ชี้ลูกทางซ้าย *)
        end;
```

< การสร้างข้อมูลแบบ binary tree >

สมมุติว่ามีข้อมูลแบบ binary tree ซึ่งมีโครงสร้างดังข้างล่าง ถ้าต้องการเพิ่ม Jim เข้าในข้อมูลที่ได้จัดลำดับไว้แล้วที่เห็นข้างล่าง ก็สามารถใส่ข้อมูลได้โดยมีหลักการดังนี้

Ann   Anna   Bill   Emy   Lisa   Tom



ก่อนอื่นต้องเปรียบเทียบลำดับของ Jim กับ Lisa ก่อน เนื่องจาก Emy อยู่ลำดับต้น จึงต้องไปทาง branch ข้างซ้าย เมื่อเปรียบเทียบกับ Anna ปรากฏว่า Jim อยู่ลำดับหลัง Anna จึงไปทางขวา เมื่อเปรียบเทียบกับ Bill ปรากฏว่าอยู่ลำดับหลัง Bill จึงไปทางขวาอีก สุดท้าย เมื่อเปรียบเทียบกับ Emy Jim มีลำดับอยู่หลัง Emy จึงไปทางขวา ปรากฏว่าเป็น nil จึงบันทึก Jim ไว้ที่ตำแหน่งดังกล่าว

ถ้ากำหนดให้ tree ชื่อ p และ node ชื่อ w การสร้างโพรซิเจอร์เรียกตัวเอง heap(p,w) ซึ่งมีโครงสร้างเป็นลักษณะเป็นต้นไม้ ก็สามารถเขียนได้เป็นดังที่เห็น

```
if node ยังว่าง
   then สร้าง node ชื่อ w ใหม่ขึ้นมา
else if node w มีค่ามากกว่า
   then เคลื่อนไปทาง branch ขวา และเรียกโพรซิเจอร์ heap()
else if node w มีค่าน้อยกว่า
   then เคลื่อนไปทาง branch ซ้าย และเรียกโพรซิเจอร์ heap()
```

เมื่อเขียนเป็นภาษา Pascal จะมีลักษณะเป็น

```
procedure heap(var p:point;w:name);
  begin
    if p=nil then
      begin
        new(p); p^.name:w; p^.left:=nil; p^.right:=nil
      end;
    else if p^.name<w then heap(p^.right,w)
    else heap(p^.left,w)
  end;
```

p เป็นพารามิเตอร์ของโพรซิเจอร์ heap ถ้าเรียกตัวเองด้วย heap(p^.right,w) หรือ heap(p^.left,w) ถ้าตำแหน่งดังกล่าวเป็น nil p ก็จะถูกเก็บค่า ณ ตำแหน่งนั้น

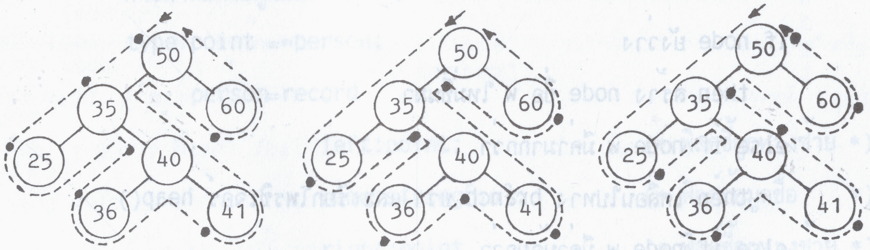
< การอ่านข้อมูลใน binary tree >

เมื่อใส่ข้อมูลแบบ binary tree ต้องมีการอ่านอย่างเป็นระเบียบ การอ่านข้อมูลนั้นจำเป็นต้องใช้การเคลื่อนที่ (traversal) ซึ่งมีอยู่ 3 แบบ คือ

1 แบบ Preorder

root ⟶ { branch ทางซ้าย / branch ทางขวา } ⟶ { branch ทางขวา / branch ทางซ้าย }

ลำดับการเคลื่อนที่เริ่มจาก root ⟶ branch ทางซ้าย ⟶ branch ทางขวา ลำดับข้อมูลที่อ่านจะเป็น 50 35 25 40 36 41 60 ดังรูป



แบบ preorder แบบ inorder แบบ postorder

2 แบบ inorder

{branch ทางซ้าย / branch ทางขวา} ⟶ root ⟶ {branch ทางขวา / branch ทางซ้าย}

ลำดับการเคลื่อนที่เริ่มจาก branch ทางซ้าย ⟶ root ⟶ branch ทางขวา

ลำดับข้อมูลที่อ่านจะเป็น 25 35 36 40 41 50 60 ซึ่งเป็นการเรียงลำดับจากน้อยไปมาก

3 แบบ postorder

root ⟶ {branch ทางซ้าย / branch ทางขวา} ⟶ {branch ทางซ้าย / branch ทางขวา}

ลำดับการเคลื่อนที่เริ่มจาก branch ทางซ้าย ⟶ branch ทางขวา ⟶ root

ลำดับข้อมูลที่อ่านจะเป็น 25 36 41 40 35 60 50

จากรูปเส้นไข่ปลาคือแนวเคลื่อนที่ จุดคือตำแหน่งอ่านข้อมูล ซึ่งจะเห็นได้ว่าการเคลื่อนที่ของทุกแบบเหมือนกัน จะต่างกันที่ตำแหน่งที่เริ่มอ่านข้อมูลตัวแรก

หลักการอ่านข้อมูลโดยทั่วไปมีดังนี้ คือ

1 เคลื่อนไปทาง branch ข้างซ้ายจนกว่าเป็น nil และกลับมาหนึ่ง node

2 เคลื่อนไปทาง branch ข้างขวาจนกว่าจะเป็น nil ระหว่างทางถ้ามีข้อมูลที่ branch ข้างซ้าย ให้ดำเนินการตามข้อ 1 และเมื่อทางขวาเป็น nil ให้กลับมาหนึ่ง node

จากหลักการดังกล่าว ถ้าเริ่มอ่านข้อมูลที่ข้อ 1 ก็เป็นแบบ preorder ถ้าเริ่มอ่านข้อมูลระหว่างข้อ 1 กับ 2 ก็เป็นแบบ inorder และถ้าเริ่มอ่านจากข้อ 2 ก็เป็นแบบ postorder

ถ้าเขียนเป็นโพรซิเจอร์ จะมีลักษณะเป็น

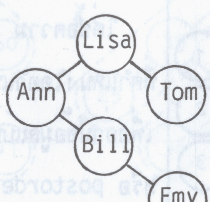
```
procedure treewalk(p:pointer);
begin
    if p<> nil then
        begin
            treewalk(p^.left);     <- (1)
            treewalk(p^.right)     <- (2)
        end                        <- (3)
end;
```

ใส่ข้อความ writeln(p^ name) ที่ตำแหน่งใดตำแหน่งหนึ่งระหว่าง 1 2 3 เพื่ออ่านข้อมูลแบบ preorder, inorder หรือ postorder

ตัวอย่างที่ 10.10 โปรแกรมโครงสร้างข้อมูลแบบ binary tree เพื่อเก็บข้อมูลและทำการเรียงลำดับ พร้อมแสดงข้อมูล

```
program tree;
type namestr=string[20];
   point=^node;
   node=record
          left:point;
          name:namestr;
          right:point
        end;
var root:point;
   girl:namestr;
   procedure heap(var p:point;w:namestr);
     begin
       if p=nil then
         begin
           new(p);
           p^.name:=w;p^.left:=nil;p^.right:=nil;
         end
       else if p^.name<w then heap(p^.right,w)
       else heap(p^.left,w)
     end;
   procedure treewalk(p:point);
     begin
       if p <> nil then
         begin
           treewalk(p^.left);
           writeln(p^.name);
           treewalk(p^.right)
         end
     end;
begin
   root:=nil;
   readln(girl);
   while girl <> '/' do
     begin
       heap(root,girl);
       readln(girl);
     end;
   treewalk(root);
end.
```

```
Lisa  [enter]
Ann   [enter]
Bill  [enter]
Tom   [enter]
Emy   [enter]
/     [enter]
Ann
Bill
Emy
Lisa
Tom
```



1 โพรซิเจอร์ heap สร้างตำแหน่งข้อมูล
2 อ่านข้อมูลทาง branch ซ้าย
3 อ่านข้อมูลทาง branch ขวา
4 โพรซิเจอร์แสดงข้อมูลแบบ inorder
5 กำหนดให้ root เป็น nil ในตอนแรก

< คำสั่งที่ใช้กับข้อมูลพอยน์เตอร์ >

ที่ผ่านมามีการใช้คำสั่ง new เพื่อจองตำแหน่งหน่วยความจำ นอกจากคำสั่ง new แล้ว ยังมีคำสั่งอื่นอีกดังที่เห็น

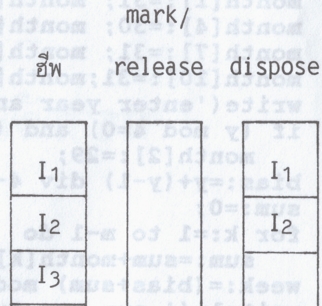
1 mark / release

คำสั่ง 2 คำสั่งนี้ต้องใช้ด้วยกัน คำสั่ง new ที่ได้กล่าวไปแล้ว เป็นคำสั่งจองตำแหน่งหน่วยความจำ ซึ่งเรียกว่าฮีพ (heap) ส่วนหน่วยความจำที่เก็บตัวแปร นิพจน์ พารามิเตอร์ ฯลฯ เรียกว่า แสตก (stack) เมื่อมีการใช้คำสั่ง new เพื่อเก็บข้อมูลมากๆ ฮีพ ก็จะใหญ่ขึ้นเรื่อยๆจนกระทั่งเข้ามาในแสตก ซึ่งจะเกิด error อย่างแน่นอน

เพื่อเป็นการป้องกันปัญหาดังกล่าว ถ้าหากไม่ต้องการใช้ตัวแปรพอยน์เตอร์แล้วก็ควรจะลบตัวแปรนั้นออกจากหน่วยความจำ สมมุติ I เป็นตัวแปรพอยน์เตอร์แบบลิสท์ (รูปที่ 1) ถ้าใช้คำสั่ง mark(I) ก่อนคำสั่ง new คำสั่งแรก และใช้คำสั่ง release(I) เมื่อไม่ต้องการใช้ตัวแปรดังกล่าว ผลจะเป็นดังรูปที่ 2

2 dispose

คำสั่งนี้มีลักษณะคล้ายคำสั่ง mark/release เมื่อใช้คำสั่ง dispose แล้ว ผลจะเป็นรูปที่ 3 กล่าวคือข้อมูลที่พอยน์เตอร์ปัจจุบันชี้อยู่ (I_3) จะถูกลบออกจากฮีพ เมื่อใช้คำสั่ง new ใหม่ก็สามารถใช้ตำแหน่งหน่วยความจำที่ถูกลบ คำสั่งนี้ห้ามใช้รวมกับ mark/release



รูปที่ 1 รูปที่ 2 รูปที่ 3

3 getmem / freemem

การดำเนินการของคำสั่งนี้มีลักษณะเหมือนคำสั่ง mark/release มาก ถ้าหากใช้คำสั่ง getmem(i,200) ก่อนคำสั่ง new และใช้คำสั่ง freemem(i,200) เมื่อไม่ต้องการใช้ตัวแปรก็จะมีการดำเนินการเช่นเดียวกับคำสั่ง mark/release 200 ก็คือ การจองหน่วยความจำ 200 ไบท์ให้กับตัวแปรพอยน์เตอร์ i ถ้าไม่จำเป็นไม่ควรใช้คำสั่งนี้ เพราะอาจเกิดปัญหาในการกำหนดเนื้อที่ของหน่วยความจำ (ฮีพ)

4 maxavail

ฟังก์ชัน ตรวจว่ามีหน่วยความจำที่สามารถใช้ได้ในขณะนี้อีกเท่าใด (หน่วยละ 16 บิท)

Coffee Break

ปฏิทินสากลที่ใช้กันอยู่ในปัจจุบันเรียกว่า gregorio เป็นปฏิทินที่พระเจ้า gregorio แห่งจักรพรรดิ์โรม ได้ทรงสร้างขึ้นในปี ค.ศ. 1582 โดยมีหลักการดังนี้

1 ให้ถือเอาวันที่ 1 มกราคม ปี 0 เป็นวันอาทิตย์

2 ปีใดที่หารด้วย 4 ลงตัวและ 100 ไม่ลงตัวให้เดือนกุมภาพันธ์ของปีนั้นมี 29 วัน

3 ปีใดที่หารด้วย 400 ลงตัวให้เดือนกุมภาพันธ์ของปีนั้นมี 29 วัน

โปรแกรมข้างล่างเป็นโปรแกรมแสดงปฏิทินของเดือนๆหนึ่งในปีที่ต้องการ

```
program gregorio;
var month:array[1..12] of integer;
   y,m,bias,k,week,day,sum:integer;
begin
   month[1]:=31; month[2]:=28; month[3]:=31;
   month[4]:=30; month[5]:=31; month[6]:=30;
   month[7]:=31; month[8]:=31; month[9]:=30;
   month[10]:=31;month[11]:=30;month[12]:=31;
   write('enter year and month ');readln(y,m);
   if (y mod 4=0) and (y mod 100<>0) or (y mod 400=0) then
      month[2]:=29;
   bias:=y+(y-1) div 4-(y-1) div 100+(y-1) div 400;
   sum:=0;
   for k:=1 to m-1 do
      sum:=sum+month[k];
   week:=(bias+sum) mod 7;
   writeln(' sun mon tue wed thu fri sat ');
   for k:=1 to 4*week do
      write(' ');
   for day:=1 to month[m] do
      begin
         write(day:4);
         if (week+day) mod 7=0 then writeln
      end
end.
```

```
enter year and month 0 1 [enter]
 sun mon tue wed thu fri sat
   1   2   3   4   5   6   7
   8   9  10  11  12  13  14
  15  16  17  18  19  20  21
  22  23  24  25  26  27  28
  29  30  31
```

แบบฝึกหัด

1 จงเขียนฟังก์ชันหาค่าตัวหารร่วมมาก โดยใช้การเรียกตัวเอง

2 คำตอบจากปัญหา 8 ควีน ถ้าหากมีการหมุนกระดานทีละ 90 องศาจนครบรอบ หรือทำการกลับหน้าและหมุนอีกจนครบรอบ จะพบว่าตำแหน่งควีนที่แสดงผลออกมาหลายๆแบบนั้น เป็นคำตอบที่ซ้ำๆกัน จงเขียนโปรแกรมหาคำตอบของปัญหา 8 ควีนที่ไม่ซ้ำกัน

3 จงใช้ข้อมูลแบบพอยน์เตอร์เขียนอินเด็กซ์ไฟล์ อินเด็กซ์ไฟล์คือ ไฟล์เก็บลำดับของข้อมูลชุดหนึ่งโดยเรียงเป็นลำดับ ข้อมูลดังกล่าวมีฟิลด์ชื่อและอายุ (เป็นแบบพอยน์เตอร์เช่นกัน) เมื่อสิ้นสุดการให้ข้อมูลให้เก็บข้อมูลดังกล่าวและอินเด็กซ์ไฟล์ลงดิสก์

บทที่ 11

กราฟฟิกใน Turbo Pascal

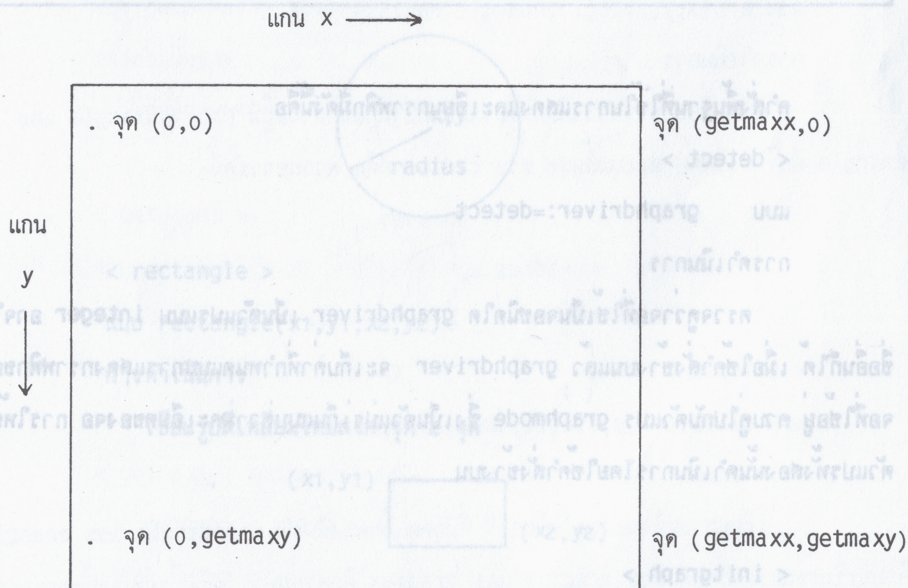
11.1 กราฟฟิกใน Turbo Pascal และคำสั่งเบื้องต้น

ใน Standard Pascal เมมีคำสั่งเขียนกราฟฟิก แต่ใน Turbo Pascal สามารถเขียนกราฟฟิกได้ โดยใช้คำสั่ง uses graph เรียก unit ชื่อ turbo.tpl หรือ graph.tpu การใช้คำสั่งดังกล่าว สามารถเขียนโดยคำสั่ง uses ดังที่เห็นข้างล่าง

```
program test_graph;
uses graph;
```

คำว่ากราฟฟิกไม่ใช่หมายถึงการเขียนกราฟเส้นตรงหรือลากเส้นเท่านั้น แต่หมายถึงการวาดรูป ยกตัวอย่างเช่น การเขียนการตูน หรือการเขียนลวดลายต่างๆ

จอของคอมพิวเตอร์จะมีลักษณะดังที่เห็นข้างล่าง นั่นคือจุดบนจอจะมีค่าเท่ากับ (0,0) ที่ตำแหน่งซ้ายบนและตำแหน่งขวาล่างมีค่าเท่ากับ (getmaxx,getmaxy)



จอ (monitor)

getmaxx และ getmaxy คือฟังกชันแบบไม่มีพารามิเตอร์ ดำเนินการส่งค่ามากสุดบนจอกราฟฟิกของแกนนอนและแกนตั้งตามลำดับ สาเหตุที่ใช้ฟังกชันทั้งสองเนื่องจาก จอแต่ละชนิดมีความละเอียดไม่เท่ากันเช่น (640,400) หรือ (720*348) เป็นต้น ฟังกชันทั้งสองจะทำการตรวจความละเอียดของจอและส่งค่าดังกล่าวกลับมา

ฟังก์ชันและโพรซิเจอร์เบื้องต้นที่ใช้สำหรับเขียนกราฟฟิกมีดังต่อไปนี้คือ

arc bar bar3d circle cleardevice clearviewport closegraph drawpoly ellipse fillellipse fillpoly floodfill getarccoords getaspectratio getfillsettings getimage getlinesettings getmaxx getmaxy getpixel gettextsettings getviewsettings getx gety imagesize initgraph line linerel lineto moverel moveto outtext outtextxy pieslice putimage putpixel rectangle sector setaspectratio setactivepage setfillpattern setfillstyle setlinestlye settextjustify settextstyle setviewport setusercharsize setvisualpage texthigh textwidth

คำสั่งพื้นฐานที่ใช้ในการแสดงและเขียนกราฟฟิกมีดังนี้คือ

< detect >

แบบ graphdriver:=detect

การดำเนินการ

ตรวจดูว่าจอที่ใช้เป็นจอชนิดใด graphdriver เป็นตัวแปรแบบ integer อาจใช้ชื่ออื่นก็ได้ เมื่อใช้คำสั่งข้างบนแล้ว graphdriver จะเก็บค่าที่กำหนดแบบการแสดงกราฟฟิกของจอที่ใช้อยู่ ควบคู่ไปกับตัวแปร graphmode ซึ่งเป็นตัวแปรเก็บแบบความละเอียดของจอ การให้ค่าตัวแปรทั้งสองนั้นดำเนินการโดยใช้คำสั่งข้างบน

< initgraph >

แบบ initgraph(graphdriver,graphmode,'a:\')

การดำเนินการ

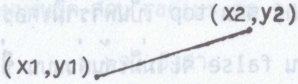
เปลี่ยนการแสดงบนจอเป็นแบบกราฟฟิก โดยนำค่า graphdriver , graphmode ที่ได้มาจากคำสั่งแรกเพื่อใช้ในการเขียนกราฟฟิกบนจอที่ใช้อยู่ ส่วน 'a:\' หมายถึงโปรแกรมที่เกี่ยวกับการแสดงกราฟฟิก (ไฟล์ graph.tpu หรือ ไฟล์.bgi) อยู่ในแผ่นดิสก์อยู่ที่ไดรฟ์ a หรืออาจเขียนเป็น '' ซึ่งจะต้องมีไฟล์ดำเนินการกราฟฟิกอยู่ที่ไดรฟ์ปัจจุบัน (Current Drive)

< line >

แบบ line(x1,y1,x2,y2)

การดำเนินการ

ลากเส้นจากจุด (x1,y1) ไปที่ (x2,y2)

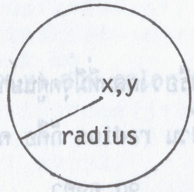


< circle >

แบบ circle(x,y,radius)

การดำเนินการ

เขียนรูปวงกลม จุดศูนย์กลางอยู่ที่ (x,y) รัศมียาว radius

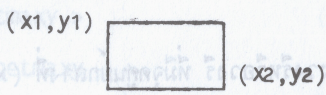


< rectangle >

แบบ rectangle(x1,y1,x2,y2)

การดำเนินการ

เขียนรูปสี่เหลี่ยมโดยส่งค่าจุด 2 จุด

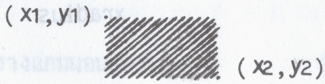


< bar >

แบบ bar(x1,y1,x2,y2)

การดำเนินการ

ระบายพื้นที่ภายในสี่เหลี่ยมโดยส่งค่าจุด 2 จุด

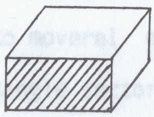


< bar3d >

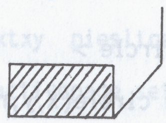
แบบ bar3d(x1,y1,x2,y2,depth,top)

การดำเนินการ

เขียนกราฟแท่ง 3 มิติ ลักษณะเป็นสี่เหลี่ยมเช่นเดียวกับ คำสั่ง bar สิ่งที่แตกต่างก็คือ การแสดงความลึก (depth) ส่วน top เป็นพารามิเตอร์แบบบูลีน ในกรณีที่เป็น true จะมีเส้นส่วนบนของกราฟแท่ง ถ้าเป็น false จะไม่มีเส้นส่วนบน ซึ่งใช้สำหรับการเขียนกราฟซ้อนๆกัน



top เป็น true



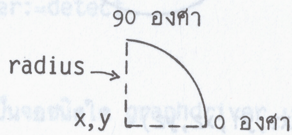
top เป็น false

< arc >

แบบ arc(x,y,stangle,enangle,radius)

การดำเนินการ

เขียนส่วนหนึ่งของวงกลมหรือวงกลมที่มีจุดศูนย์กลางอยู่ที่ (x,y) stangle และ enangle คือ มุมเริ่มต้น และมุมสิ้นสุด ส่วน radius ก็คือ ความยาวรัศมี

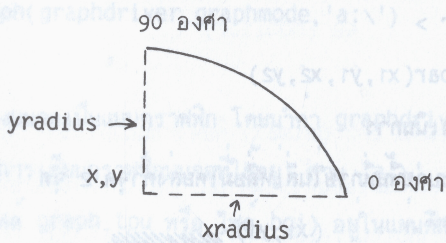


< ellipse >

แบบ ellipse(x,y,stagle,enangle,xradius,yradius)

การดำเนินการ

เขียนส่วนหนึ่งของวงรีหรือวงรี ที่มีจุดศูนย์กลางที่ (x,y) stangle และ enangle คือมุมเริ่มต้นและมุมสิ้นสุด ส่วน xradius และ yradius คือความยาวรัศมีทางแกนนอนและแกนตั้ง

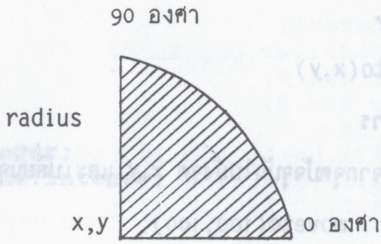


< pieslice >

แบบ pieslice(x,y,stangle,enangle,radius)

การดำเนินการ

เขียนและระบายส่วนหนึ่งของวงกลมหรือวงกลมที่มีจุดศูนย์กลางที่ (x,y) stangle และ enangle คือ มุมเริ่มต้นและมุมสิ้นสุด ส่วน radius คือ ความยาวรัศมี



< putpixel >

แบบ putpixel(x,y,color)

การดำเนินการ

เขียนจุดบนจอที่จุด (x,y) ส่วน color นั้นเป็นคำสั่งกำหนดสี

< getpixel >

แบบ getPixel(x,y)

การดำเนินการ

ส่งสีที่จุด x,y ว่าเป็นสีอะไร

< getmaxx , getmaxy >

แบบ getmaxx , getmaxy

การดำเนินการ

ส่งค่าสูงสุดทางแกนนอนและแกนตั้งของจอที่ใช้งานอยู่

< getx , gety >

แบบ getx , gety

การดำเนินการ

ส่งค่าจุดปัจจุบันทางแกนนอนและแกนตั้ง

< moveto >

แบบ moveto(x,y)

การดำเนินการ

เลื่อนจุดปัจจุบันไปยังจุด x,y

< lineto >

แบบ lineto(x,y)

การดำเนินการ

ลากเส้นจากจุดปัจจุบันไปยังจุด x,y และเปลี่ยนจุดปัจจุบันไปที่จุด x,y

```
moveto(100,100);
lineto(200,200);
```

มีผลเหมือนกับ line(100,100,200,200); moveto(200,200);

< linerel >

แบบ linerel(x,y);

การดำเนินการ

ลากเส้นจากจุดปัจจุบันไปตามแกนนอนและแกนตั้ง จำนวน x จุด และ y จุด สมมุติจุดปัจจุบันคือ (5,5) คำสั่ง linerel(10,10) ก็คือการลากเส้นจากจุด (5,5) ไปที่ (15,15)

< moverel >

แบบ moverel(x,y)

การดำเนินการ

เลื่อนจุดปัจจุบันไปตามแกนนอนและแกนตั้ง จำนวน x จุดและ y จุด สมมุติว่าปัจจุบันอยู่ที่ (5,5) คำสั่ง moverel(10,10) คือการเปลี่ยนจุดปัจจุบันจากจุด (5,5) ไปที่ (15,15)

< cleardevice >

แบบ clerdevice

การดำเนินการ

สบภาพกราฟฟิกจากจอ

< closegraph >

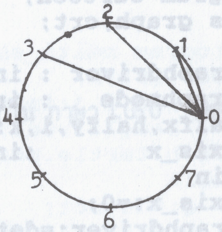
แบบ closegraph

การดำเนินการ

ออกจากการแสดงกราฟฟิก (ใช้เมื่อเลิกการแสดง)

ตัวอย่างที่ 11.1 กราฟฟิกรูปเพชร

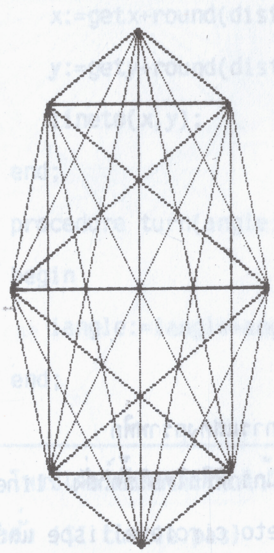
< วิธีคิด >



แบ่งวงกลมออกเป็น 8 ส่วน

และลากเส้นระหว่างจุดทั้ง 8 จุด

```
program diamond;
uses graph,crt;
const n=8;
var rd:real;
    k,j,x1,y1,x2,y2:integer;
    graphdriver,graphmode:integer;
begin
  rd:=pi/180;
  graphdriver:=detect;
  initgraph(graphdriver,graphmode,'');.......................... 1
  for k:=0 to n-2 do
     begin
       x1:=round(100*cos(k*360/n*rd))+getmaxx div 2;
       y1:=round(100*sin(k*360/n*rd))+getmaxy div 2; ....... 2
       for j:=k+1 to n-1 do
          begin
            x2:=round(100*cos(j*360/n*rd))+getmaxx div 2;
            y2:=round(100*sin(j*360/n*rd))+getmaxy div 2;.. 3
            line(x1,y1,x2,y2)                            ...... 4
          end
     end;
  repeat until keypressed;
  closegraph                                        ...................... 5
end.
```



1 เริ่มแสดงกราฟฟิก

2 กำหนดรัศมียาว 100 ทำการเปลี่ยนจุดเริ่มต้น k จาก 0 ถึง 6 โดยมีจุดศูนย์ที่ (getmaxx div 2,getmaxy div 2)

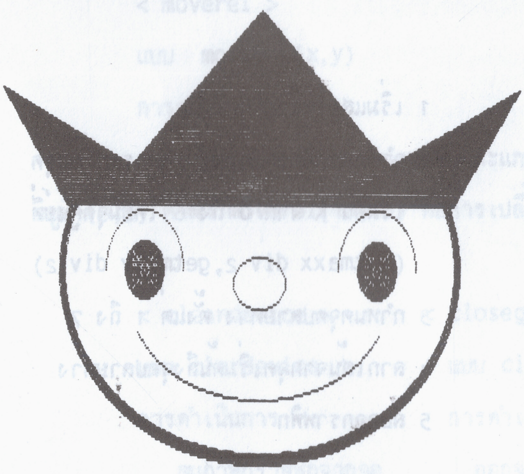
3 กำหนดจุดปลายทาง ตั้งแต่ 1 ถึง 7

4 ลากเส้นจากจุดเริ่มต้นถึงจุดปลายทาง

5 ลบจอกราฟฟิก

ตัวอย่างที่ 11.2 กราฟฟิกรูปการ์ตูน

```
program cartoon;
uses graph,crt;
var
  graphdriver : integer;  { The Graphics device driver }
  graphmode   : integer;  { The Graphics mode value }
  halfx,halfy,i,k: integer;
  axis_x       :integer;
begin
  axis_x:=0;
  graphdriver:=detect;
  Initgraph(graphdriver,graphmode, '');                   ..................... 1
  halfx:=getmaxx div 2;
  halfy:=getmaxy div 2;
  for i:=1 to 5 do
     ellipse(halfx,halfy,160,20,150+i,70+i);
  ellipse(270,halfy,0,180,30,23);
  ellipse(450,halfy,0,180,30,23);
  for i:=1 to 2 do
     begin
        for k:=1 to 15 do
           circle(270+axis_x,halfy,k);
        axis_x:=axis_x+180
     end;
  line(halfx-150,halfy-25,halfx+150,halfy-25);
  moveto(halfx-150,halfy-25);
  lineto(halfx-200,halfy-70);
  lineto(halfx-100,halfy-40);
  lineto(halfx,halfy-100);
  lineto(halfx+100,halfy-40);                              ..................... 2
  lineto(halfx+200,halfy-70);
  lineto(halfx+150,halfy-25);
  floodfill(halfx,halfy-30,1);
  ellipse(halfx,halfy+5,0,360,20,10);
  ellipse(halfx,halfy+10,200,340,100,40);
  repeat until keypressed;
  closegraph;
end.
```



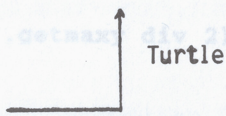
1 เริ่มการแสดงกราฟฟิก

2 เขียนกราฟฟิกโดยใช้คำสั่ง line lineto moveto circle ellispe และ fillpoly

11.2 Turtle Graphics และ Recusive Graphics

< Turtle Graphics >

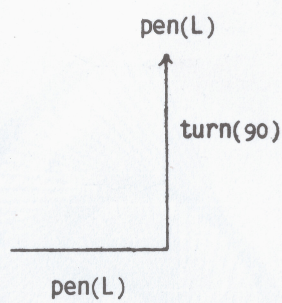
Turtle Graphics เป็นการเขียนกราฟฟิก ที่นิยมใช้กันมากใน ภาษา LOGO โดยเขียนกราฟฟิกด้วยการกำหนดความยาวและมุมเพื่อเขียนเส้น



ลักษณะพิเศษของ Turtle Graphics คือ กำหนดความยาวและมุมเพื่อทำการเขียนเส้น ดังนั้น จึงต้องสร้างโพรซิเจอร์ pen และ turn เพื่อหาตำแหน่งจุดและลากจากจุดปัจจุบันไปที่ตำแหน่งที่ต้องการ และเปลี่ยนมุมปัจจุบันซึ่งสามารถเขียนได้ดังที่เห็น

var iangle:real; (* ให้ iangle เป็นตัวแปร global แทนมุมปัจจุบัน *)

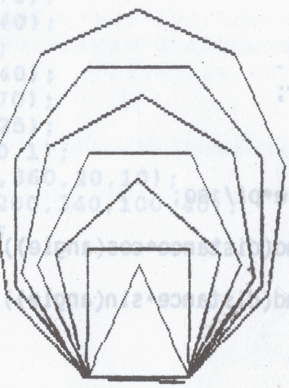
```
procedure pen(distance:integer);
var angle:real;
   x,y:integer;
begin
   angle:=iangle*pi/180;
   x:=getx+round(distance*cos(angle));
   y:=gety+round(distance*sin(angle));
   lineto(x,y);
end;
procedure turn(angle:real);
begin
   iangle:=iangle+angle;
end;
```



โพรซิเจอร์ทั้งสองจะเก็บไว้ในไฟล์ชื่อ turtle.pas เมื่อต้องการใช้จะเรียกไฟล์ดังกล่าว โดยใช้คำสั่ง {$I turtle.pas}

ตัวอย่างที่ 11.3 การเขียนรูปหลายเหลี่ยม (จาก 3 ถึง 9) ด้านเท่า

```
program multi_shapes;
uses graph;
var g,h,i,j:integer;
    iangle:real;
{$i b:turtle.pas }                          .................. 1
begin
   g:=detect;
   initgraph(g,h,'');
   for g:=3 to 9 do
      begin
        iangle:=0;
        moveto(getmaxx div 2,getmaxy div 2);
        for h:=0 to g do                          ........ 2
           begin
              pen(50);                            ........ 3
              turn(-360 div g)                    ........ 4
           end;
      end;
      readln;
   closegraph
end.
```



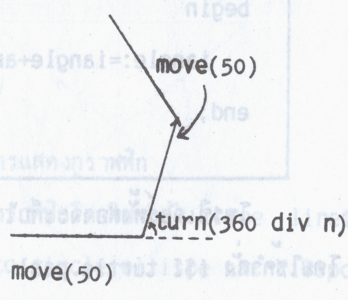
1 เรียกโพรซิเจอร์ turtle.pas

2 เขียนรูป g เหลี่ยมด้านเท่า

3 ลากเส้นยาว 50 ไปทิศทาง

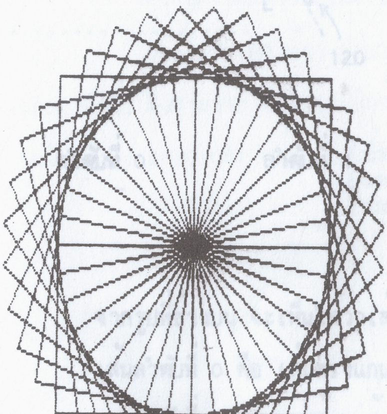
4 มุมปัจจุบัน (iangle)

เพิ่มมุมปัจจุบันไป (360/n) องศา



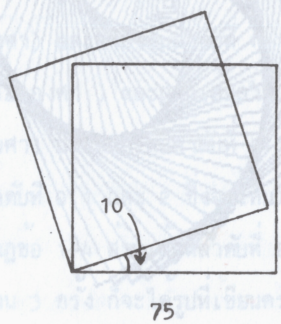
ตัวอย่างที่ 11.4 การวาดรูปสี่เหลี่ยมจตุรัส และเปลี่ยนมุมที่ละ 10 องศาจนครบรอบ

```
program multi_square;
uses graph;
var g,h,i,j:integer;
     iangle:real;
{$i b:turtle.pas }
begin
   g:=detect;
   initgraph(g,h,'');
   moveto(getmaxx div 2 ,getmaxy div 2);
   iangle:=0;
   for i:=1 to 36 do
       begin
          for j:=1 to 4 do
             begin
                pen(75);               ........... 1
                turn(90)
             end;
          turn(10)                     ........... 2
       end;
  closegraph
end.
```



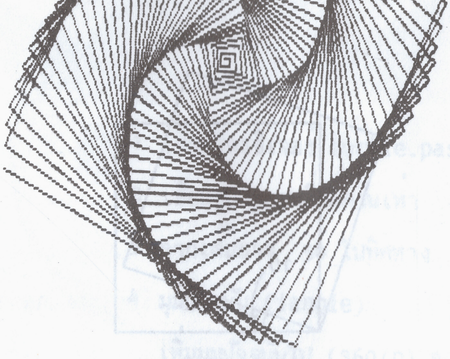
1 ลากสี่เหลี่ยมจตุรัสความยาวด้านละ 75

2 เปลี่ยนมุม 10 องศา



ตัวอย่างที่ 11.5 รูปน้ำวน

```
program timetunnel;
uses graph,crt;
var i,j,length:integer;
    iangle:real;
{$I b:turtle.pas}
begin
   i:=detect;
   initgraph(i,j,'');
   moveto(getmaxx div 2,getmaxy div 2);
   length:=3;
   while length<160 do          ............. 1
     begin
        pen(length);
        turn(89);               ............. 2
        length:=length+1        ............. 3
     end;
  repeat until keypressed;
  closegraph;
end.
```



1 วนซ้ำจนกว่า ความยาวเท่ากับ 160

2 ลากเส้นยาวเท่ากับ length ไปทางมุมที่หักจากมุมปัจจุบัน 89 องศา

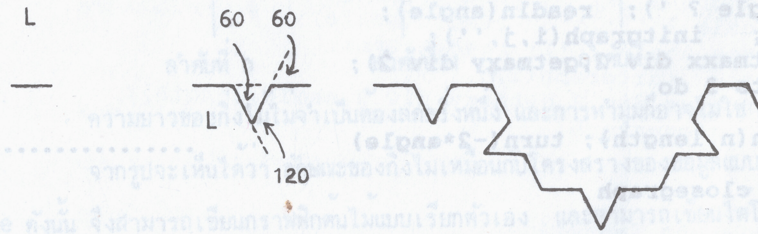
3 เพิ่มความยาว length ขึ้น 1

< Recursive Graphics >

คอมพิวเตอร์กราฟฟิกในปัจจุบัน การเขียนกราฟฟิกแบบเรียกตัวเอง (Recursive Graphics) เป็นวิธีการเขียนกราฟฟิกที่ต่างไปจากการเขียนกราฟฟิกทั่วไป กล่าวคือ กราฟฟิกทั่วไปเป็นการกำหนดตำแหน่งบนจอ และเขียนกราฟฟิก ซึ่งการเขียนกราฟฟิกดังกล่าวไม่สามารถเขียนกราฟฟิกรูปต้นไม้หรือรูปธรรมชาติอื่น แต่กราฟฟิกแบบเรียกตัวเองเป็นวิธีการเขียนที่สามารถเขียนกราฟฟิกได้ใกล้ธรรมชาติมาก

< การเขียนเส้น koch โดยวิธีเรียกตัวเอง >

เส้น koch มีลักษณะดังที่เห็น



ลำดับที่ 0 ลำดับที่ 1 ลำดับที่ 2

angle เท่ากับ 60

จากรูปที่ข้างบน จะเห็นได้ว่าเส้น koch มีลักษณะดังนี้คือ

เส้นลำดับที่ 0 คือ เส้นตรงแกนนอน

เส้นลำดับที่ n ประกอบด้วยเส้นลำดับที่ n-1 สี่เส้น โดยมีหลักการดังนี้ คือ

1 ลากเส้นลำดับที่ n-1

2 บวกมุมปัจจุบันไปอีก angle (60 องศา) และลากเส้นลำดับที่ n-1

3 บวกมุมปัจจุบันไปอีก -2*angle (120องศา) และลากเส้นลำดับที่ n-1

4 บวกมุมปัจจุบันไปอีก angle (60 องศา) และลากเส้นลำดับที่ n-1

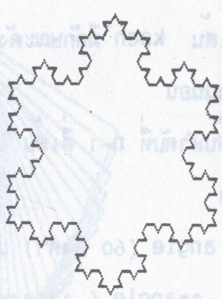
จากรูปข้างบนเป็นการเขียนตัวอย่างของเส้นลำดับที่ 0 1 และ 2 ซึ่งจะเห็นได้ว่า เส้นลำดับที่ 1 ประกอบไปด้วยเส้นลำดับที่ 0 สี่เส้นเขียนตามกฎข้อ 1-4 ส่วนเส้นที่ลำดับที่ 2 ประกอบด้วยลำดับที่ 1 สี่เส้นเรียงกันถ้าหากทำการวนซ้ำเป็นจำนวน 3 ครั้ง ก็จะได้รูปที่เขียนครบหนึ่งรอบ

ตัวอย่างที่ 11.7 การเขียนเส้น koch โดยใช้กราฟฟิกแบบเรียกตัวเอง

```
program koch_shape;
uses graph;
var i,j,k,n,length,angle:integer;
    iangle:real;
{$I b:turtle.pas }
   procedure koch(n,length:integer);
     begin
       if n=0 then pen(length) ........................................ 1
              else
               begin
                  koch(n-1,length);turn(angle);
                  koch(n-1,length);turn(-2*angle);               ............. 2
                  koch(n-1,length);turn(angle);
                  koch(n-1,length)
               end
     end;
begin { main }
   write('order ? ');  readln(n);
   write('length ? '); readln(length);
   write('angle ?');   readln(angle);
   i:=detect;   initgraph(i,j,'');
   moveto(getmaxx div 2,getmaxy div 2);
   for k:=1 to 3 do
     begin
        koch(n,length); turn(-2*angle)                           ...................... 3
     end;
   readln;   closegraph
end.
```

```
order    1 [enter]            order   3 [enter]
length  30 [enter]            length  4 [enter]
angle   60 [enter]            angle  60 [enter]
```





1 ลากเส้นลำดับที่ 0 ไปทางมุมและยาวเท่ากับค่าที่กำหนดให้

2 เรียกตัวเองลำดับที่ n-1

3 เขียนเส้นลำดับที่ n จำนวน 3 รอบ

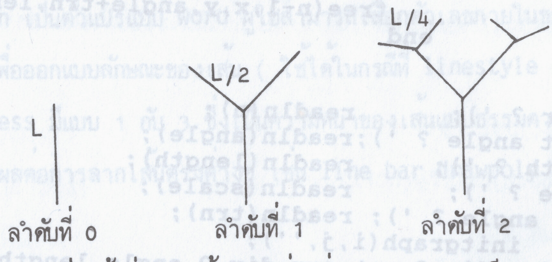
< การเขียนต้นไม้โดยใช้การเรียกตัวเอง >

การเขียนรูปต้นไม้มีวิธีการเขียนดังนี้ คือ

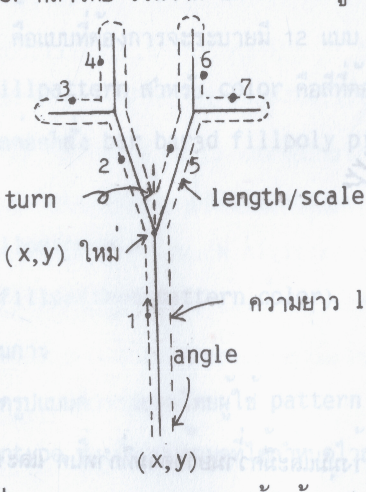
1 เส้นลำดับที่ 0 คือ เส้นตั้งฉากยาว L

2 เส้นลำดับที่ 1 คือ เส้น 2 เส้นที่เกิดจากปลายเส้นลำดับที่ 0 ทำมุมระหว่างกันเป็นมุมฉาก โดยความยาวของเส้นทั้ง 2 คือ L/2

3 เส้นลำดับที่ 2 คือ เส้น 4 เส้นที่เกิดจากปลายเส้นลำดับที่ 1 ทำมุมระหว่างกันเป็นมุมฉาก โดยความยาวของเส้นทั้ง 2 คือ L/4



ความยาวของกิ่งไม้ไม่จำเป็นต้องลดครึ่งหนึ่ง และการทำมุมก็อาจไม่ใช่ 90 องศาก็ได้ จากรูปจะเห็นได้ว่า ลักษณะของกิ่งไม้เหมือนกับโครงสร้างของข้อมูลแบบ binary-tree ดังนั้น จึงสามารถเขียนกราฟฟิกต้นไม้แบบเรียกตัวเอง และสามารถเขียนได้โดยใช้หลักการเคลื่อนที่แบบ preorder กล่าวคือ เริ่มจาก แม่ ⟶ ลูกทางซ้าย ⟶ ลูกทางขวา



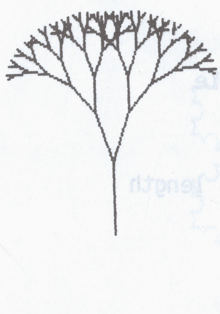
จะเห็นได้ว่า การเรียกตัวเองของรูปต้นไม้จำเป็นต้องส่งพารามิเตอร์ดังต่อไปนี้คือ ลำดับ ตำแหน่ง (x,y) ใหม่ มุมที่ต้องการให้กิ่งต่อไปทำมุมกับกิ่งปัจจุบัน และความยาวของกิ่งต่อไป

ตัวอย่างที่ 11.7 การวาดรูปต้นไม้โดยใช้การเรียกตัวเอง

```
program tree_curve;
uses graph;
var i,j,y,xx,yy,n,length,angle,trn:integer;
   scale,iangle:real;
{$I b:turtle.pas }
  procedure tree(n,x,y,angle:integer;length:real);
    begin
      if n<>0 then begin
                     moveto(x,y);
                     iangle:=angle;
                     pen(round(length));        .................. 1
                     x:=getx;y:=gety;
                     tree(n-1,x,y,angle-trn,length/scale); ... 2
                     tree(n-1,x,y,angle+trn,length/scale); ... 3
                   end

     end;
 begin
   write('order ? ');        readln(n);
   write('start angle ? ');readln(angle);
   write('length ? ');      readln(length);
   write('scale ? ');       readln(scale);
   write('tree angle ? '); readln(trn);
   i:=detect;   initgraph(i,j,'');
   tree(n,getmaxx div 2,getmaxy div 2,angle,length);
   readln;   closegraph
 end.
```

```
order 7 [enter]
start angle -90 [enter]
length 30 [enter]
scale 1.4 [enter]
tree angle 20 [enter]
```




1 ลากเส้นลำดับที่ 0 ไปทางมุมและมีความยาวตามที่กำหนด และเปลี่ยนจุดเริ่มต้นใหม่

2 ลากเส้นลำดับที่ n-1 ไปทางขวา

3 ลากเส้นลำดับที่ n-1 ไปทางซ้าย

รูปทางขวามือ กำหนดให้ scale ของกิ่งไม้ทางขวามีค่าน้อยกว่าทางซ้าย

11.8 คำสั่งกราฟฟิกที่ใช้ในการกำหนดแบบการเขียนกราฟฟิก

คำสั่งกราฟฟิกที่ใช้ในการกำหนดแบบการเขียนเส้น ระบายและการเขียนตัวอักษรมีดังนี้

< setlinestyle >

แบบ setlinestyle(linestyle,pattern,thickness)

การดำเนินการ

กำหนดรูปแบบเส้นที่ต้องการลาก (linestyle) แบบ (pattern) และความหนา (thickness) โดยมีหลักการดังนี้คือ

linestyle มีแบบตั้งแต่ 0 ถึง 4 (0 ถึง 3 เป็นแบบที่มีอยู่ 4 เป็นแบบที่ผู้ใช้กำหนด)

pattern เป็นตัวแปรแบบ word ผู้ใช้สามารถเลือกตัวเลขภายในขอบเขต $0..$ffff หรือ 0..65535 เพื่อออกแบบลักษณะของเส้น (ใช้ได้ในกรณีที่ linestyle เป็น 4 เท่านั้น)

thickness มีแบบ 1 กับ 3 ซึ่งเป็นความหนาของเส้นแบบธรรมดาและแบบหนา

คำสั่งนี้มีผลต่อการลากเส้นตรงต่างๆ เช่น line bar drawpoly rectangle เป็นต้น

< setfillstyle >

แบบ setfillstyle(pattern,Color)

การดำเนินการ

กำหนดรูปแบบ (pattern) และสี (color) ที่จะระบาย

pattern คือแบบที่ต้องการจะระบายมี 12 แบบ แบบสุดท้ายคือแบบที่ผู้ใช้สามารถกำหนดเองโดยคำสั่ง setfillpattern สำหรับ color คือสีที่ต้องการระบาย

คำสั่งนี้มีผลต่อคำสั่ง bar bar3d fillpoly pieslice floodfill

< setfillpattern >

แบบ setfillpattern(pattern,color)

การดำเนินการ

กำหนดรูปแบบการระบายโดยผู้ใช้ pattern โดยกำหนดให้ pattern เป็นตัวแปรแบบ fillpatterntype ซึ่งเป็นแบบข้อมูลที่ได้กำหนดไว้ล่วงหน้าแล้วมีลักษณะเป็น

fillpatterntype :array [1..8] of byte;

การให้ค่ากับ pattern อาจให้ค่าได้มากน้อยขึ้นอยู่ตามความต้องการ ดังตัวอย่าง

const pattern:fillpatterntype=($aa,$55,$aa,$55,$aa,$55,$aa,$55)

< settextstyle >

แบบ settextstyle(font,direction,size);

การดำเนินการ

กำหนดแบบตัวอักษร (font) ทิศตั้งหรือนอน (direction) และขนาด (size)

font คือแบบตัวอักษรมีตั้งแต่ 0 ถึง 4

direction คือแนวตัวอักษร 0 คือการเขียนแนวนอน 1 คือการเขียนแนวตั้ง

size คือขนาดตัวอักษร ตั้งแต่ 1 ถึง 10 (จากเล็กไปใหญ่)

คำสั่งนี้มีผลต่อการเขียนหรือตรวจตัวอักษรบนจอกราฟฟิก เช่น outtext texthigh ในแผ่นดิสก turbo pascal ต้องมีไฟล์ .chr ด้วย ถ้าไม่มีจะเป็นตัวอักษรธรรมดา

< settextjustify >

แบบ settextjustify(horiz,vert);

การดำเนินการ

กำหนดตำแหน่งที่ต้องการแสดงอักษร horiz และ vert ต่างก็มีค่าตั้งแต่ 0 ถึง 2

horiz มีค่าเป็น 0 หมายถึงแสดงชิดทางซ้าย 1 หมายถึงตรงกลางและ 2 หมายถึงชิดขวา

vert มีค่าเป็น 0 หมายถึงแสดงชิดทางล่าง 1 หมายถึงตรงกลางและ 2 หมายถึงชิดบน

< outtext,outtextxy >

แบบ outtext(textstring),outtextxy(x,y,textstring)

การดำเนินการ

คำสั่ง outtext เป็นเขียน textstring (ข้อมูลแบบสตริง) ลักษณะ ขนาด และแนวตัวอักษรนั้นกำหนดโดยใช้คำสั่ง settextstyle ตำแหน่งที่เริ่มเขียนคือ ตำแหน่งปัจจุบัน เนื่องจากการเขียนตัวอักษรบนจอกราฟฟิกไม่สามารถใช้คำสั่ง write หรือ writeln ได้ นอกจากคำสั่ง outtext แล้ว ยังมีคำสั่งouttextxy ซึ่งมีแบบเป็น outtextxy(x,y,textstring) x และ y ก็คือ ตำแหน่งที่ต้องการแสดงตัวอักษร

ex

```
settextstyle(1,0,5);    settextjustify(2,0);
outtextxy(getmaxx,getmaxy,'Oh! God');
```

การแสดงตัวอักษรก็จะเป็นแบบแนวนอนขนาดกลาง และอยู่ตัวสุดท้าย (d) อยู่ชิดขวาล่าง

< drawpoly >

แบบ drawpoly(numpoint,polypoints)

การดำเนินการ

ลากเส้นจำนวน numpoint เส้น โดยลากจากจุดที่ 1 ไปยังจุดที่ 2 จากจุดที่ 2 ไป จนกระทั่งถึงจุดสุดท้าย polypoints เป็นแบบข้อมูลที่มีลักษณะเป็นดังข้างล่าง

```
type polypoints = array [1..numpoint] of record
                                              x : integer;
                                              y : integer;
                                            end;
```

< fillpoly >

แบบ fillpoly

การดำเนินการ

ระบายพื้นที่ภายในรูปหลายเหลี่ยมที่ลากโดยคำสั่ง drawpoly ในกรณีที่เส้นแรกกับเส้นสุดท้ายไม่บรรจบกัน การระบายจะระบายภายในโดยให้จุดแรกกับจุดสุดท้ายเชื่อมต่อกัน

< floodfill >

แบบ floodfill(X,Y ,Color:word)

การดำเนินการ

ระบายพื้นที่ที่ตำแหน่ง (x,y) และรอบๆข้างโดยสี color ถ้าหากตำแหน่ง (x,y) อยู่ภายรูป จะเป็นการระบายภายในรูป ถ้าตำแหน่งอยู่นอกรูประบายพื้นที่รอบนอก

< setusercharsize >

แบบ setusercharsize(multx,divx,multy,divy)

การดำเนินการ

ปรับขนาดตัวอักษรที่ใช้อยู่ในปัจจุบันให้ใหญ่เล็กตามความต้องการ เช่น ถ้าต้องการให้ตัวอักษรกว้างขึ้นกว่าปัจจุบันเป็นสองเท่าและสูงเป็นครึ่งหนึ่ง ก็สามารถเขียนได้โดยใช้คำสั่ง

setusercharsize(2,1,1,2)

< fillellipse >

แบบ fillellipse(x,y,xradius,yradius)

การดำเนินการ

เขียนและระบายวงรี โดยมีจุดศูนย์กลางวงรีอยู่ที่ (x,y) และความยาวรัศมีแกน x และแกน y เท่ากับ xradius และ yradius ตามลำดับ

< sector >

แบบ sector(x,y,stangle,endangle,xradius,yradius)

การดำเนินการ

ดำเนินการเช่นเดียวกับคำสั่ง fillellipse และยังสามารถกำหนดมุมเริ่มต้น และมุมสิ้นสุดที่พารามิเตอร์ stangle,endangle ตามลำดับ

< getLinesettings >

แบบ getlinesettings(lineinfo)

การดำเนินการ

ส่งลักษณะของเส้น ซึ่งถูกกำหนดโดยคำสั่ง setlinestyle ล่าสุดให้กับ lineinfo ซึ่งเป็นข้อมูลแบบ linesettingstype โดยมีโครงสร้างดังที่เห็นข้างล่าง

```
type linesettingstype = record
                             linestyle,pattern,thickness:word
                        end;
```

< getfillsettings >

แบบ getfillsettings(fillinfo);

การดำเนินการ

ส่งสีและลักษณะของการระบาย ซึ่งถูกกำหนดโดยคำสั่ง setfillstyle ล่าสุดให้กับตัวแปร fillinfo ซึ่งมีแบบเป็น fillsettingstype โดยมีโครงสร้างดังข้างล่าง

```
type    fillsettingstype = record
                              pattern,color:word
                           end;
```

< gettextsettings >

แบบ gettextsettings(textinfo);

การดำเนินการ

ส่งชนิดตัวอักษร ซึ่งกำหนดโดยคำสั่ง settextstyle ล่าสุดให้กับ textinfo ซึ่งเป็นข้อมูลแบบ textsettingstype โดยมีโครงสร้างดังข้างล่าง

```
type  charsizetype = 1..10;
   textsettingstype = record
                          font,direction : word;
                          charsize       : charsizetype;
                          horiz,vert     : word
                          end;
```

horiz และ vert คือ ค่าตำแหน่งที่เริ่มเขียนตัวอักษรทางแกน x และ y ตามลำดับ

< setviewport >

แบบ setviewport(X_1,Y_1,X_2,Y_2,clip);

การดำเนินการ

สร้างขอบเขตขึ้นภายกรอบสี่เหลี่ยม ซึ่งมีตำแหน่งมุมซ้ายบนที่ (X_1,Y_1) และมุมขวาล่างที่ (X_2,Y_2) สมมุติว่าใช้คำสั่ง setviewport(100,100,200,200,true) เมื่อใช้คำสั่ง line(0,0,20,20) เส้นที่ปรากฏจะเป็นเส้นที่ลากจากตำแหน่ง (100,100) ถึง (120,120) สำหรับ clip เป็นข้อมูลแบบบูลีน ถ้าเป็น true การแสดงกราฟฟิกจะไม่เลยกรอบที่ได้กำหนดขึ้นมา

< getviewsetting >

แบบ getviewsettings(viewport);

การดำเนินการ

ส่งค่าของกรอบที่กำหนดโดยคำสั่ง setviewport ล่าสุดให้กับ viewport ซึ่งเป็นตัวแปรแบบ viewporttype โดยมีโครงสร้างดังข้างล่าง

```
type viewporttype = record
                  x1,y1,x2,y2:word;  clip:boolen
              end;
```

< clearviewport >

แบบ clearviewport

การดำเนินการ

ลบกราฟฟิกภายในกรอบที่กำหนดโดยคำสั่ง sertviewport ในกรณีที่ต้องการกำหนดกรอบขึ้นใหม่ ต้องใช้คำสั่ง setviewport ใหม่อีก

< getimage >

แบบ getimage(X_1,Y_1,X_2,Y_2,Bitmap);

การดำเนินการ

เก็บภาพบนจอที่อยู่ในกรอบสี่เหลี่ยม (x_1,y_1,x_2,y_2) ลงบัฟเฟอร์ Bitmap ซึ่งเป็นตัวแปรแบบ pointer (แบบตัวแปรนี้มีอยู่ใน Turbo Pascal) ชื่อตัวแปรอาจใช้ชื่ออื่นได้ คำสั่งนี้เหมาะสำหรับการเขียนกราฟฟิกให้เคลื่อนไหว โดยเก็บรูปต่างๆไว้ที่ตัวแปรที่ต่างกัน และแสดงออกเป็นลำดับ

< putimage >

แบบ putimage(x,y,bitmap,bitbit)

การดำเนินการ

แสดงกราฟฟิกที่เก็บไว้ที่ bitmap จากคำสั่ง getimage โดยมุมบนซ้ายของรูปอยู่ที่ตำแหน่ง (x,y) bitbit คือ แบบการแสดงขอบเขต (1 ถึง 5) รายละเอียดมีดังนี้คือ

1 แสดงภาพปรกติ

2 แสดงภาพ โดยตรวจดูว่าตำแหน่งที่จะแสดงภาพว่ามีการแสดงภาพอยู่ก่อนแล้วหรือไม่ การแสดงภาพจะแสดงในกรณีที่ตำแหน่งดังกล่าวไม่มีการแสดงภาพอยู่ก่อน (xor)

3 แสดงภาพ โดยตรวจดูว่าตำแหน่งที่จะแสดงภาพว่ามีการแสดงภาพอยู่ก่อนแล้วหรือไม่ การแสดงภาพจะแสดงทุกกรณีไม่ว่าตำแหน่งดังกล่าวมีการแสดงภาพก่อนหรือไม่ (or)

4 แสดงภาพ โดยตรวจดูว่าตำแหน่งที่จะแสดงภาพว่ามีการแสดงภาพอยู่ก่อนแล้วหรือไม่ การแสดงภาพจะในกรณีที่ตำแหน่งดังกล่าวมีการแสดงภาพอยู่ก่อน (and)

5 แสดงภาพ ซึ่งกลับข้างกับการแสดงภาพที่มีอยู่ก่อน (not)

< imagesize >

แบบ imagesize(X_1, Y_1, X_2, Y_2 : word)

การดำเนินการ

ฟังก์ชันหาค่าจำนวนไบท์ ที่เก็บส่วนของจอภาพในกรอบสี่เหลี่ยม (X_1, Y_1, X_2, Y_2)

< getarccoords >

แบบ getarccoords(arccoords)

การดำเนินการ

ส่งพารามิเตอร์ของคำสั่ง arc ล่าสุดให้กับตัวแปร arccoords ซึ่งมีแบบข้อมูลเป็น arccoordstyle โดยมีโครงสร้างเป็นข้อมูลดังข้างล่าง endx,eny คือตำแหน่งของปลายเส้นโค้ง

```
type arccoordstype = record
                        x,y:integer;
                        startx,starty,endx,endy:word
                     end;
```

< textheight,textwidth >

แบบ textheight(textstring),textwidth(textstring)

การดำเนินการ

ส่งค่าความสูงและความกว้างของสตริง textstring โดยหน่วยเท่ากับจำนวนจุดบนจอ คำสั่งนี้มีประโยชน์ต่อการจัดการตำแหน่งของตัวอักษรบนจอให้เหมาะสม

< setaspectratio >

แบบ setaspectratio(xasp,yasp)

การดำเนินการ

กำหนดอัตราส่วนของการแสดงกราฟฟิกทางแกนนอนและแกนตั้ง คำสั่งนี้มีประโยชน์ในการวาดวงกลมให้กลมจริงๆ ในกรณีที่คำสั่ง circle ไม่สามารถวาดวงกลมได้ วิธีการกำหนดค่านั้นสามารถเขียนได้ดังนี้

setaspectratio(50,60); ในกรณีที่ต้องการให้วงกลมกว้างขึ้น

และ setaspectratio(60,50); ในกรณีที่ต้องการให้วงกลมสูงขึ้น

< getaspectratio >

แบบ getaspectratio(xasp,yasp)

การดำเนินการ

กสรแสดงกราฟฟิกออกจอ สมมุติว่าใช้คำสั่ง rectangle(0,0,50,50) การเขียนกราฟฟิกจะไม่เป็นรูปสี่เหลี่ยมจตุรัส คำสั่ง getaspectratio ใช้สำหรับส่งอัตราส่วนแกน x กับ y โดยให้ xasp กับ yasp เป็นตัวแปรแบบ word และ rate เป็นตัวแปรแบบ real เมื่อต้องการเขียนรูปสี่เหลี่ยมจตุรัส หรือลากเส้นแทยงมุมบนจอก็สามารถเขียนดังข้างล่าง

ex

```
getaspectratio(xasp,yasp);
rate:=xasp/yasp;
rectangle(0,0,50,round(50*rate));
```

หรือ line(0,0,100,round(100*rate));

< setactivepage >

แบบ setactivepage(page)

การดำเนินการ

เก็บรูปกราฟฟิกลงในจอที่ต้องการ ดังตัวอย่างข้างล่าง

ex

```
setactivepage(1);
rectangle(0,0,100,100);
```

จากตัวอย่าง รูปสี่เหลี่ยมจะถูกเก็บอยู่ที่จอ 1 (โดยปรกติจอที่ใช้อยู่คือ จอที่ 0)

< setvisualpage >

แบบ setvisualpage(page)

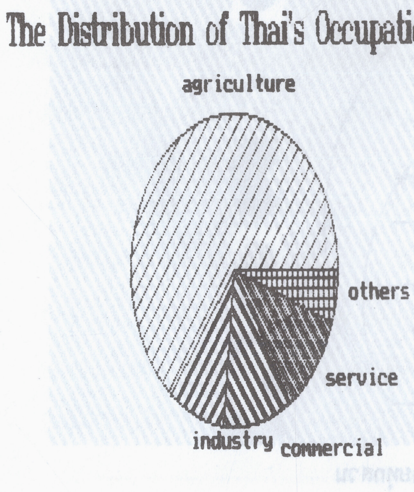
การดำเนินการ

แสดงกราฟฟิกที่เก็บไว้โดยคำสั่ง setactivepage

จากตัวอย่างคำสั่ง setactivepage เมื่อใช้คำสั่ง setvisualpage(1) รูปสี่เหลี่ยมที่เก็บไว้ที่จอ 1 จะปรากฏบนจอ คำสั่งทั้งสองเป็นคำสั่งในการเลือกจอและแสดงกราฟฟิกของจอนั้นๆ ซึ่งเหมาะที่จะแสดงรูปกราฟฟิกที่แตกต่างกันโดยทำการเรียกจอสลับไปมา

ตัวอย่างที่ 11.8 กราฟพิกวงกลมแสดง อาชีพของประชาชนไทย

```
program statistics;
uses graph;
type occu=(agriculture,industry,commercial,service,others);
var i,j,stangle,angle,dumy,x,y,z:integer;
   a:array[agriculture..others] of real;
   b:array[agriculture..others] of string[15];
   c:occu;   total,postx,posty:real;
begin
   a[agriculture]:=17555000; b[agriculture]:= 'agriculture';
   a[industry]:=    2406000; b[industry]:=    'industry';
   a[commercial]:=  2956000; b[commercial]:= '  commercial'; ... 1
   a[service]:=     2908000; b[service]:=     'service';
   a[others]:=      1453000; b[others]:=       'others';
   total:=27278000;
   stangle:=0;
   i:=detect; initgraph(i,j,'');
   x:=getmaxx div 2;y:=getmaxy div 2;
   settextjustify(1,2);   settextstyle(1,0,1);
   outtextxy(x,70,'The Distribution of Thai''s Occupation');
   for c:=agriculture to others do
     begin
        setfillstyle(ord(c)+3,1);
        angle:=stangle+round(a[c]/total*360);                ....... 2
        pieslice(x,y,stangle,angle,80);                       ..... 3
        dumy:=(stangle+angle) div 2;
        postx:=x+90*cos(dumy*pi/180);
        posty:=y-75*sin(dumy*pi/180);                    .......... 4
        settextstyle(0,0,1); settextjustify(0,0);
        outtextxy(round(postx),round(posty),b[c]);          ....... 5
        stangle:=angle;
     end;
   readln;    closegraph
end.
```



1 ข้อมูลอาชีพ

2 คำนวณหามุมเพื่อวาดส่วนของวงกลม

3 เขียนกราฟพิกวงกลม

4 คำนวณตำแหน่งเพื่อเขียนอาชีพ

5 แสดงอาชีพ

ตัวอย่างที่ 11.9 กราฟฟิกรูป โดเรมอน

```
Program Doramon_Animation;
uses graph,crt;
Var  i,j,X1,X2,Y1,Y2,XC,YC:Integer;
     ArcCoordinate0, ArcCoordinate1 : ArcCoordsType;
Begin
   i:=detect;initgraph(i,j,'');
   XC:=getmaxx div 2-20;YC:=getmaxy div 2-25; { center points }
   Ellipse(XC,YC,0,230,YC+3,YC-57);           { face }
   GetArcCoords(ArcCoordinate0);
   Ellipse(XC,YC,310,360,YC+3,YC-57);         { head }
   GetArcCoords(ArcCoordinate1);
   With ArcCoordinate0 do
     begin MoveTO(Xend,Yend); X1:=Xend;Y1:=Yend end; { neck }
   With ArcCoordinate1 do
     begin LineTo(Xstart,Ystart); X2:=Xstart; Y2:=Ystart end;
   Ellipse(XC,YC+30,0,65,YC-27,YC-75);        { rigth cheek }
   Ellipse(XC,YC+30,330,360,YC-27,YC-75);
   Ellipse(XC,YC+30,112,210,YC-27,YC-75);     { left cheek }
   Ellipse(XC-28,YC-60,0,360,28,25);          { left eye }
   Ellipse(XC+28,YC-60,0,360,28,25);          { rigth eye }
   Ellipse(XC-18,YC-50,0,180,7,4);            { left pupil }
   Ellipse(XC+18,YC-50,0,150,5,4);            { rigth pupil }
   Circle(XC,YC-33,14);                       { nose }
   Line(XC,YC-23,XC,YC+45);                   { upper mouth }
   Ellipse(XC+70,YC-10,60,180,50,8);          { upper mustache }
   Ellipse(XC-70,YC-10,0,120,55,8);
   Ellipse(XC+74,YC+5,60,180,55,5);           { center mustache }
   Ellipse(XC-74,YC+5,0,120,50,5);
   Line(XC+20,YC+15,XC+100,YC+26);            { lower mustache }
   Line(XC-20,YC+15,XC-100,YC+25);
   Ellipse(XC,YC+27,230,312,120,24);          { mount }
   Ellipse(XC,YC+15,215,325,100,50);
   floodfill(xc,yc+30,2);
   repeat until keypressed;
   Closegraph
End.
```

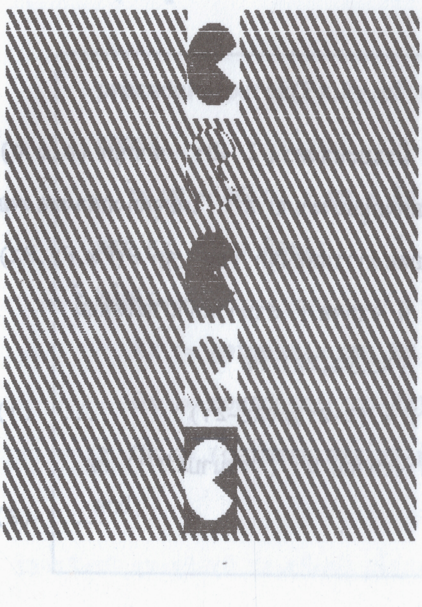


การวาดส่วนต่างๆ ได้มีการใช้คอมเมนต์อธิบายทุกส่วน

ตัวอย่างที่ 11.10 กราฟฟิกรูป pucman โดยใช้คำสั่ง putimage แสดงออกจอทั้ง 5 แบบ

```
program pucman;
uses graph,crt;
var graphdriver,graphmode:integer;
    size:word;
    puc:pointer;
    center_x,center_y:integer;
    i,distance:integer;
begin

graphdriver:=detect;
initgraph(graphdriver,graphmode,'a:\');
pieslice(20,20,30,330,20);                            ............... 1
size:=imagesize(0,0,40,40);
getmem(puc,size);
center_y:=getmaxy div 2;
getimage(0,0,40,40,puc^);
distance:=200;
  repeat
     setfillstyle(5,1);                                .............. 2
     bar(200,0,getmaxx-200,center_y+30);
     for i:=0 to 4 do
        putimage(distance,0+i*40,puc^,i);              .............. 3
     distance:=distance+20;
   until keypressed or (distance> getmaxx div 2);
repeat until keypressed;
closegraph;
freemem(puc,100);
end.
```



1 เขียนรูปกราฟฟิกรูป pucman อ้าปาก

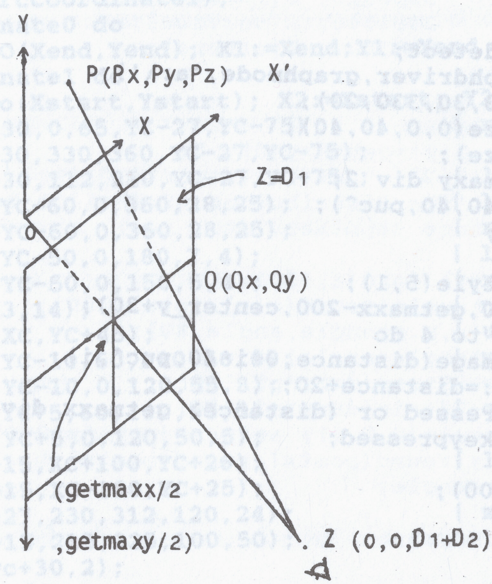
2 ระบายสีพื้นโดยคำสั่ง setfillstyle

3 ให้ pucman ขยับตัวภายในกรอบจากซ้ายไปขวา

11.4 กราฟฟิกสามมิติ

< การแปลงค่าสามมิติลงจอ >

การเขียนกราฟฟิกสามมิตินั้น ต้องทำการแปลงค่าจากตำแหน่งสามมิติมาที่จอ หลักการแปลงค่าสามารถคำนวณได้จากรูปข้างล่าง



จากรูป จอภาพอยู่ขนานกับระนาบ X,Y มีระยะทางห่าง $Z = D_1$ จากจุด (0,0,0) สำหรับตากำหนดให้ตั้งอยู่บนจุด $(0,0,D_1+D_2)$ และจอตั้งอยู่บนระนาบ X'Y' โดยจุดกึ่งกลางของจอกับจุด (0,0,0) และตาตั้งอยู่บนเส้นตรงเดียวกัน เมื่อต้องการแสดงจุด เช่น จุด P(Px,Py,Pz) บนจอก็จะปรากฏเป็นจุด Q(Qx,Qy) ซึ่งหลักการหาจุด Q จากจุด P สามารถคำนวณได้ดังนี้

เส้นตรงที่ลากผ่านจุด (x_1,y_1,z_1) และ (x_2,y_2,z_2) คือ

$(x-x_1)/(x_2-x_1) = (y-y_1)/(y_2-y_1) = (z-z_1)/(z_2-z_1)$ ..... (1)

ระนาบที่ขนานกับระนาบ X,Y และระยะทางเท่ากับ D_1 (ระนาบX'Y') คือ

$z=D_1$ ..... (2)

เมื่อนำตำแหน่งวัตถุ (Px,Py,Pz) แทน (X_2,Y_2,Z_2) ตำแหน่งตา (0,0,D_1+D_2) แทน (X_1,Y_1,Z_1) จะได้ค่า x และ y เป็นดังข้างล่าง

$x = (D_2/(D_1+D_2-P_z))*P_x$

..... (3)

$y = (D_2/(D_1+D_2-P_z))*P_y$

ค่า x,y ที่คำนวณได้เป็นค่าบนระนาบ X'Y' เมื่อแปลงเป็นค่า Qx,Qy จะได้เป็น

$Qx = (D_2/(D_1+D_2-P_z))*Px*KX+getmaxx\ div\ 2$

..... (4)

$Qy = getmaxy\ div\ 2 -(D_2/(D_1+D_2-P_z))*Py*KY$

ค่า KX และ KY คือ ค่าคงที่ซึ่งขึ้นอยู่กับความละเอียดของจอ ในกรณีนี้จะใช้จอขนาด 26*20 เซนติเมตร ค่า KX และ KY จะเท่ากับ

KX = getmaxx div 26 ; KY = getmaxy div 20 ..... (5)

เมื่อให้ระยะทาง $D_1 = D_2 = 50$ เซนติเมตร ค่า Qx,Qy จะเท่ากับ

Qx = (50/(100-Pz)*Px)*getmaxx/26+getmaxx div 2

..... (6)

Qy = getmaxy-(50/(100-pz))*py*getmaxy/20

เมื่อเขียนเป็นโพรซิเจอร์ แปลงจุด P เป็น Q จะมีลักษณะเป็นดังที่เห็นข้างล่าง โดยที่ขนาดของวัตถุจะอยู่บนจอ เมื่อตำแหน่ง x,y,z มีค่าอยู่ภายใน -20 ถึง +20

```
procedure purse(Px,Py,pZ:real;Var Qx,Qy:integer);
const
   x_cm = 26.0; y_cm = 20.0;
var kx, ky : Real;
begin
   kx := getmaxx/x_cm; ky := getmaxy/y_cm;
   Qx := round(50/(100-Pz)*Px*kx)+(getmaxx div 2);
   Qy := (getmaxy div 2)-round(50/(100-Pz)*Py*ky)
end;
```

< การเคลื่อนที่วัตถุสามมิติ (Transformaton) >

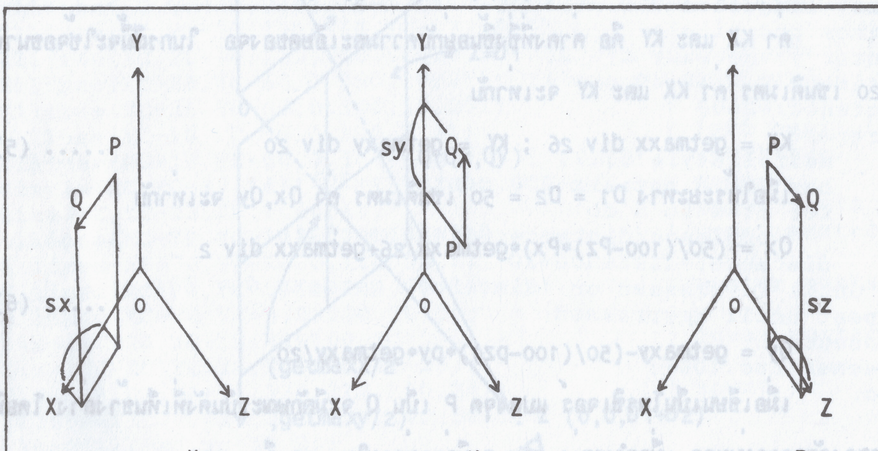
การเคลื่อนที่วัตถุสามมิตินั้น นอกจากการเคลื่อนที่ในทางเส้นตรงแล้ว การเคลื่อนที่รอบแกนก็นับเป็นสิ่งจำเป็นอีกด้วย การเคลื่อนที่วัตถุสามมิติ (Transformation) มีอยู่ 2 วิธีคือ

1 การเคลื่อนที่เป็นเส้นตรง (Translation)

2 การเคลื่อนที่รอบแกน (Rotation)

รายละเอียดมีดังนี้คือ

1 การเคลื่อนที่เป็นเส้นตรง



(ก) แกน X (ข) แกน Y (ค) แกน Z

1.1 การเคลื่อนที่ทางแกน X

การเคลื่อนที่ทางแกน X จาก P(Px,Py,Pz) ไปเป็นระยะทาง sx ไปที่ Q(Qx,Qy,Qz) ตำแหน่ง Q จะเป็น Qx=Px+sx, Qy=Py, Qz=Pz ดังรูป (ก) ..... (7)

1.2 การเคลื่อนที่ทางแกน Y

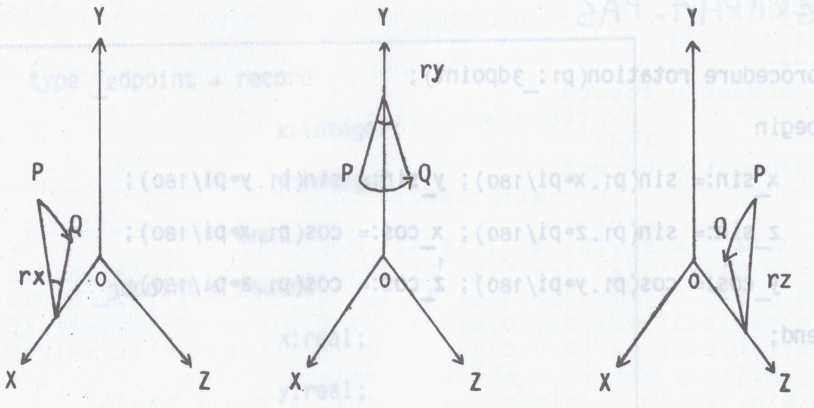
การเคลื่อนที่ทางแกน X จาก P(Px,Py,Pz) ไปเป็นระยะทาง sy ไปที่ Q(Qx,Qy,Qz) ตำแหน่ง Q จะเป็น Qx=Px, Qy=Py+sy, Qz=Pz ดังรูป (ข) ..... (8)

1.3 การเคลื่อนที่ทางแกน Z

การเคลื่อนที่ทางแกน Z จาก P(Px,Py,Pz) ไปเป็นระยะทาง sx ไปที่ Q(Qx,Qy,Qz) ตำแหน่ง Q จะเป็น Qx=Px, Qy=Py, Qz=Pz+sz ดังรูป (ค) ..... (9)

เมื่อรวมสูตร (7),(8),(9) ตำแหน่ง Q จะเป็น Qx=Px+sx,Qy=Py+sy,Qz=Pz+sz

2 การหมุนรอบแกน (Rotation)



(ก) แกน X (ข) แกน Y (ค) แกน Z

2.1 การหมุนรอบแกน X

การหมุนรอบแกน X คือ การหมุนตำแหน่ง P(Px,Py,Pz) ไปรอบแกน X ดังรูป (ก) ซึ่งใช้ฟังก์ชันมาตรฐาน sin และ cos เพื่อทำการคำนวณ ตำแหน่ง Q(Qx,Qy,Qz) จะเป็น

Qx = Px , Qy = Py*cos(rx)-Pz*sin(rx) , Qz = Py*sin(rx)+Pz*cos(rx) ..... (10)

2.2 การหมุนรอบแกน Y

การหมุนรอบแกน Y คือ การหมุนตำแหน่ง P(Px,Py,Pz) ไปรอบแกน Y ดังรูป (ข) เมื่อทำการคำนวณ ตำแหน่ง Q(Qx,Qy,Qz) จะเป็น

Qx = Pz*sin(ry)+Px*cos(ry) , Qy = Py , Qz = Pz*cos(ry)-Px*sin(ry) ..... (11)

2.3 การหมุนรอบแกน Z

การหมุนรอบแกน Z คือ การหมุนตำแหน่ง P(Px,Py,Pz) ไปรอบแกน Z ดังรูป (ค) เมื่อทำการคำนวณ ตำแหน่ง Q(Qx,Qy,Qz) จะเป็น

Qx = Px*cos(rz)-Pz*sin(rz) , Qy = Px*sin(rz)+Py*cos(rz) , Qz = Pz ..... (12)

เมื่อเขียนเป็นโพรซิเจอร์รวมกับโพรซิเจอร์แรก จะมีลักษณะเป็นดังที่เห็นข้างล่าง

GRAPH4.PAS

```
procedure rotation(p1:_3dpoint);
begin
   x_sin:= sin(p1.x*pi/180); y_sin:= sin(p1.y*pi/180);
   z_sin:= sin(p1.z*pi/180); x_cos:= cos(p1.x*pi/180);
   y_cos:= cos(p1.y*pi/180); z_cos:= cos(p1.z*pi/180)
end;

procedure trans(p1:_3dpoint;var p2:_3dpoint);
Var a, b, c : _3dpoint;
begin
   a.x:=p1.x;a.y:=x_cos*p1.y-x_sin*p1.z;a.z:=x_sin*p1.y+x_cos*p1.z;
   b.x:=y_sin*a.z+y_cos*a.x; b.y:=a.y; b.z:=y_cos*a.z-y_sin*a.x;
   c.x:=z_cos*b.x-z_sin*b.y; c.y:=z_sin*b.x+z_cos*b.y; c.z := b.z;
   p2.x:=c.x+x_shift;    p2.y:=c.y+y_shift; p2.z:=c.z+z_shift
end;

procedure purse(p1:_3dpoint; Var p2:_2dpoint);
const d1=50.0; d2=50.0;
Var   kx, ky : Real;
begin
   kx:=getmaxx/26; ky:=getmaxy/20;
   p2.x:=round(d2/(d1+d2-p1.z)*p1.x*kx)+(getmaxx div 2);
   p2.y:=(getmaxy div 2)-round(d2/(d1+d2-p1.z)*p1.y*ky)
end;
```

แบบข้อมูล _2dpoint และ _3dpoint เป็นแบบข้อมูลที่กำหนดขึ้นโดยมีโครงสร้างดังข้างล่าง (แบบข้อมูลนี้ต้องกำหนดที่ต้นโปรแกรมทุกครั้งที่มีการใช้กราฟฟิกสามมิติ)

```
type _2dpoint = record
                    x:integer;
                    y:integer
                end;
     _3dpoint = record
                    x:real;
                    y:real;
                    z:real
                end;
```

ตัวแปร x_cos,y_cos,z_cos x_sin,y_sin,z_sin,x_shift,y_shift และ z_shift เป็นตัวแปรแบบเลขจำนวนจริงที่ต้องประกาศเป็นตัวแปรแบบ global ที่ต้นโปรแกรม

หนังสือเล่มนี้จะเก็บ โพรซิเจอร์ทั้ง 3 ไว้ในไฟล์ graph4 โปรแกรมในตัวอย่างต่อไปจะดำเนินการเรียกไฟล์โดยใช้คำสั่งเรียกมาใช้ดังที่เห็น {$I graph4.pas} ผู้ใช้อาจสร้างเป็น unit เพื่อความเร็วในการดำเนินการก็ได้ สำหรับการส่งพารามิเตอร์นั้น จะส่งเป็นเรคอร์ดเพื่อความสะดวก จึงได้ทำการกำหนดแบบข้อมูล _2dpointและ_3dpoint และส่งพารามิเตอร์เป็นแบบเรคอร์ด

การเก็บข้อมูลของตำแหน่งจุดนั้น ใช้วิธีเก็บตำแหน่งจุดเป็นค่าคงที่แบบเรคอร์ดที่เห็น

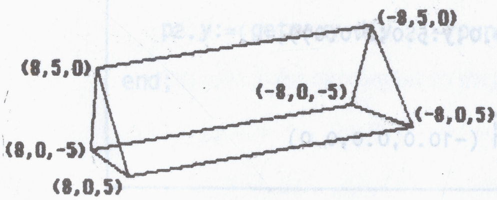
```
const   data : array[1..3,1..2] of _3dpoint =
          (
          ((x:-10.0;y:5.0;z:0.0),(x:-10.0;y:0.0;z:0.0)),
          ((x:-10.0;y:0.0;z:0.0) (x:-10.0;y:0.0;z:5.0)),
          ((x:-10.0;y:0.0;z:5.0),(x:-10.0;y:5.0;z:0.0)),
          );
```

ตัวอย่างเช่น data[1,2] มีตำแหน่งอยู่ที่ (-10.0,0.0,0.0)

ตัวอย่างที่ 11.11 กราฟฟิกรูปท่อนไม้

WOOD.PAS

```
program test_3d;
uses graph;
type _2dpoint = record
                  x:integer; y:integer
               end;
     _3dpoint = record
                  x:real;y:real;z:real
               end;
const  data : array[1..4,1..4] of _3dpoint  =
(((x:-8;y:5;z:0),(x: 8;y:5;z: 0),(x: 8;y:0;z: 5),(x:-8;y:0;z: 5)),
 ((x:-8;y:5;z:0),(x: 8;y:5;z: 0),(x: 8;y:0;z:-5),(x:-8;y:0;z:-5)),
 ((x:-8;y:0;z:5),(x: 8;y:0;z: 5),(x: 8;y:0;z:-5),(x:-8;y:0;z:-5)),...1
 ((x:-8;y:0;z:5),(x:-8;y:0;z:-5),(x:-8;y:5;z: 0),(x:-8;y:0;z: 5)));
var  x_shift, y_shift, z_shift : real;
     x_sin, y_sin, z_sin,x_cos, y_cos, z_cos : real;
     i,j:integer;
     c:_3dpoint;
     d:array[1..4,1..4] of _2dpoint;
{$i b:graph4.pas }
begin
   i:=detect;
   initgraph(i,j,'');
   x_shift:=0.0; y_shift:=0.0; z_shift:=0.0;
   c.x:=10.0;    c.y:=15.0;    c.z:=10.0;                  .................. 2
   rotation(c);                                            .................. 3
   for i := 1 to 4 do
     begin
       for j:=1 to 4 do
         begin
           trans(data[i,j],data[i,j]);
           purse(data[i,j],d[i,j]);
         end;
       drawpoly(4,d[i]);                                   .................. 4
     end;
   readln;
   closegraph
end.
```



1 กำหนดข้อมูลของท่อนไม้

2 ให้ค่าเริ่มต้นแก่ c และ shift

3 หมุนรอบแกน x y z 10 15 และ 10 องศา

4 ลากเส้นท่อนไม้โดยคำสั่ง drawpoly

< การส่งข้อมูลของรูปโดยใช้คีย์บอร์ด 1 ถึง 9 >

การวาดรูปโดยส่งตำแหน่งข้อมูลทีละตำแหน่งเป็นวิธีที่ล่าช้า วิธีส่งข้อมูลตำแหน่งของวัตถุอาจใช้วิธีลากเส้นจากคีย์บอร์ด และทำการหมุนไปรอบๆแกน Y โพรซิเจอร์ cross ที่เห็นเป็นโพรซิเจอร์ที่แสดงตำแหน่งปัจจุบันโดยใช้เครื่องหมาย +

DRAW.PAS 1 of 2

```
Procedure Cross(x,y : Integer);
Var i : Integer;
  t : Boolean;
begin
   if ((x <= getmaxx) and (x >= 0)) and
   ((y >= 0) and (y <= getmaxy)) then                  1
   begin
     for i := -3 to 3 do
        If (getpixel(x+i,y) <> 0) then putpixel(x+i,y,0)    2
        else putpixel(x+i,y,1);
     for i := -2 to 2 do
        If (getpixel(x,y+i) <> 0)then putpixel(x,y+i,0)     3
        else putpixel(x,y+i,1);
   end
end;
```

โพรซิเจอร์ cross เป็นโพรซิเจอร์ทำการแสดงเครื่องหมาย + ที่ตำแหน่ง X,Y ซึ่งเป็นพารามิเตอร์ เมื่อใช้คำสั่ง cross(x,y) ก็จะปรากฏ + บนจอ การดำเนินการมีดังนี้คือ

1 ตรวจดูว่าตำแหน่ง X,Y ตกจอหรือไม่ ถ้าไม่ตกจอให้ดำเนินการ ข้อ 2 และข้อ 3

2 ถ้าตำแหน่ง X จากซ้ายและขวา 3 จุดไม่มีการแสดงกราฟฟิกให้เขียน 7 จุดทางแกน X

3 ถ้าตำแหน่ง Y จากบนและล่าง 3 จุดไม่มีการแสดงกราฟฟิกให้เขียน 5 จุดทางแกน Y

โพรซิเจอร์ที่ต้องใช้อีกโพรซิเจอร์หนึ่ง คือโพรซิเจอร์ picture

DRAW.PAS 282

```
procedure picture(var data:datatype;var L:integer);
var i : char;
    j,k,cx,cy,step,x_old,y_old:integer;
    draw:boolean;
begin
  x_shift := 0.0; y_shift := 0.0; z_shift := 0.0;
  draw := false;
  for j:=1 to 100 do with data[0,j] do
   begin
     x := 0.0; y := 0.0; z := 0.0
   end;
  j := detect; initgraph(j,k,'');
  line(getmaxx div 2,0,getmaxx div 2,getmaxy);
  line(0,getmaxy div 2,getmaxx,getmaxy div 2);
  cx := 50; cy := 50; step := 5;
  x_old := cx; y_old := cy;
  cross(cx,cy);
  i := readkey;
  L := 1;
  while i <> '5' do
   begin
     i := readkey;
     case i of
      '1' : begin cx:=cx-step; cy:=cy+step; end;
      '2' :       cy:=cy+step;
      '3' : begin cx:=cx+step; cy:=cy+step; end;
      '4' :       cx:=cx-step;
      '6' : begin cx:=cx+step; end;
      '7' : begin cx:=cx-step; cy:=cy-step; end;
      '8' :       cy:=cy-step;
      '9' : begin cx:=cx+step; cy:=cy-step; end;
      '+' : step:=step+1;          '-' : step:=step-1;
      'D','d':if not draw then draw:=true else draw:=false;
      'S','s':begin
                data[0,L].x := 10-(getmaxx-cx)/getmaxx*20;
                data[0,L].y := (getmaxy-cy)/getmaxy*30-20;
                L := L+1;
              end;
     end;
     if cx<0 then cx:=getmaxx else if cx>getmaxx then cx:=0;
     if cy<0 then cy:=getmaxy else if cy>getmaxy then cy:=0;
     if step < 1 then step := 1
     else if step > 128 then step := 128;
     cross(x_old,y_old);
     if draw then setcolor(1) else setcolor(0);
     line(cx,cy,x_old,y_old);
     x_old := cx; y_old := cy;
     cross(cx,cy);
   end;
   cleardevice
end;
```

โพรซิเจอร์ picture เป็นโพรซิเจอร์ที่มี 2 พารามิเตอร์ คือ data และ point data เป็นพารามิเตอร์แบบ datatype (array [1..100] of _3dpoint) เก็บตำแหน่งของจุดที่ผู้ใช้ต้องการเขียนกราฟฟิก (โดยการกด s,S) ส่วน point นั้นเป็นพารามิเตอร์จำนวนเต็มเก็บจำนวนเส้นรอบวัตถุ การดำเนินการของโพรซิเจอร์ picture คือ

1 เริ่มจากการลากเส้นกึ่งกลาง

2 กำหนดตำแหน่งแรกที่(50,50)และให้ step(ระยะทางการเคลื่อนที่ของตัว +)เป็น 2

3 cx และ cy คือ ตำแหน่งปัจจุบัน X old และ y old คือ ตำแหน่งที่แล้ว

4 อ่านค่าจากผู้ใช้ว่าต้องการลากเส้นไปทิศใดโดยเลข (1,2,3,4,6,7,8,9) ในกรณีที่ต้องการเก็บข้อมูลที่ตำแหน่งดังกล่าว ให้กด s หรือ S ตำแหน่งดังกล่าวก็จะถูกเก็บลงที่ตัวแปรdata ถ้าต้องการลากเส้นระหว่างจุดด้วยก็ให้กด d หรือ D เมื่อเขียนภาพเรียบร้อยแล้วกด 5

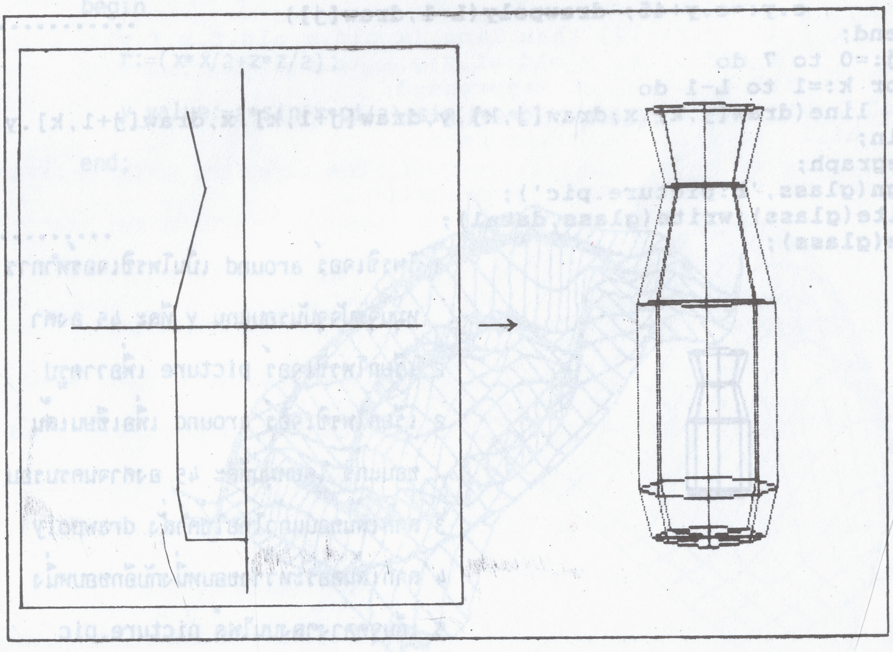
5 ในกรณีที่กด + หรือ – หมายถึงการเพิ่มหรือลดค่า step

6 ตรวจดูว่าตำแหน่งปัจจุบันตกจอหรือไม่ถ้าตกจอให้เคลื่อนกลับมาที่ 0

7 ตรวจดูว่าถ้า draw เป็น true ให้สีของเส้น setcolor(1) ถ้า drawเป็นfalse ให้สีของเส้นเป็น setcolor(0) (สีเดียวกับสีพื้นของจอ)

ให้โพรซิเจอร์ cross และ picture อยู่ในไฟล์ draw.pic

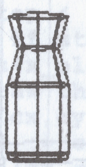
รูปที่ลากสมมุติว่าลากเป็นดังรูปที่ 1 เมื่อหมุนรอบแกน y ก็จะได้รูปเป็นสามมิติดังที่เห็น



GLASS.PAS

ตัวอย่างที่ 11.12 การวาดรูปแก้วสามมิติโดยใช้โพรซิเจอร์ cross และ picture

```
program drawing;
uses graph,crt;
type  2dpoint=record
              x:integer; y:integer
           end;
   _3dpoint=record
              x:real;y:real;z:real
           end;
   datatype = array[0..8,1..10] of _3dpoint;
var data ,data1:datatype;
    x_sin,y_sin z_sin,x_cos,y_cos,z_cos,x_shift,y_shift,z_shift:Real;
    draw : array[0..8,1..100] of _2dpoint;
    j,k,l:integer;    c:_3dpoint;
    glass:file of datatype;
   procedure around(angle : Real; p1 :_3dpoint; Var p2 : _3dpoint); ..1
     begin
       p2.x  := sin(angle*pi/180)*p1.z+cos(angle*pi/180)*p1.x;
       p2.y  := p1.y;
       p2.z  := cos(angle*pi/180)*p1.z-sin(angle*pi/180)*p1.x
     end;
{$i b:graph4.pas }{$I b:draw.pic}
begin { main }
   picture(data,L);  ............................................ 2
   x_shift:=0;y_shift:=0;z_shift:=0;c.x:=0;c.y:=0;c.z:=0;
   for j:=0 to 8 do
     begin
         for k:=1 to L-1 do
           begin
              around(c.y,data[0,k],data1[j,k]); ..................... 3
              purse(data1[j,k],draw[j,k])
           end;
           c.y:=c.y+45;  drawpoly(L-1,draw[j])  ................ 4
     end;
  for j:=0 to 7 do
     for k:=1 to L-1 do
        line(draw[j,k].x,draw[j,k].y,draw[j+1,k].x,draw[j+1,k].y); ... 5
  readln;
  closegraph;
  assign(glass,'b:picture.pic');
  rewrite(glass);write(glass,data1);
  close(glass);                                         ............... 6
end.
```



1 โพรซิเจอร์ around เป็นโพรซิเจอร์ทำการหมุนจุดปัจจุบันรอบแกน y ทีละ 45 องศา

2 เรียกโพรซิเจอร์ picture เพื่อวาดรูป

2 เรียกโพรซิเจอร์ around เพื่อเขียนเส้นขอบแก้ว โดยหมุนทีละ 45 องศาจนครบรอบ

3 ลากเส้นขอบแก้วโดยใช้คำสั่ง drawpoly

4 ลากเส้นต่อระหว่างขอบหนึ่งกับอีกขอบหนึ่ง

5 เก็บจุดต่างๆลงบนไฟล์ picture.pic

< การเขียนกราฟฟิกสามมิติแสดงฟังก์ชั่น >

การแสดงรูปกราฟฟิกของฟังก์ชั่น นอกจากการแสดงแบบ y:=sin(x) ซึ่งเป็นฟังก์ชั่นที่มีตัวแปรเพียงตัวเดียวแล้ว ยังสามารถแสดงกราฟฟิกฟังก์ชั่นที่มีตัวแปร 2 ตัวได้อีกด้วย

สมมติว่าตำแหน่งจุด p คือ P(Px,Py,Pz) โดยที่ Py เป็นฟังก์ชั่นของ Px และ Pz P(Px,Py,Pz) อาจเขียนใหม่เป็น P(Px,Py,Pz) = P(Px,f(Px,Pz),Pz)

ฟังก์ชั่นดังกล่าว อาจเป็นฟังก์ชั่นใดๆก็ได้ หลักการลากเส้นมีดังนี้คือ

1 เนื่องจากตำแหน่งที่อยู่ภายในจอคือ -20 ถึง 20 จึงกำหนดให้ค่า z เริ่มต้นที่ -12

2 เปลี่ยนค่า x ระหว่าง -12 ถึง 12 โดยเพิ่มทีละ 1 เมื่อเพิ่ม x แล้วทำการคำนวณหาค่า y และลากเส้นต่อระหว่างค่า y ที่แล้วกับค่าที่คำนวณได้

3 เมื่อการดำเนินการข้อ 2 สิ้นสุด จึงเพิ่มค่า z อีก 1 และดำเนินการข้อ 2

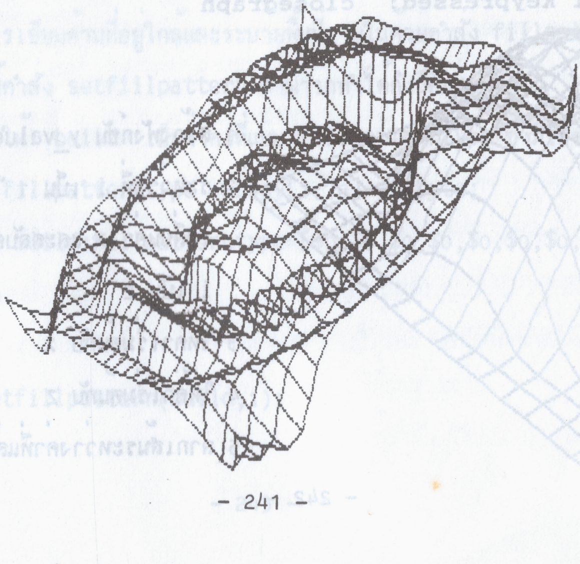
4 การดำเนินการจะดำเนินการจน z เท่ากับ 12

5 ดำเนินการข้อ 1 ถึง 4 โดยสลับตัวแปร x กับ z

การดำเนินการจากข้อ 1 ถึง 5 อาจจะเพิ่มค่า x กับ z ทีละ 0.5 หรือ 2 ก็ได้

ตัวอย่างของฟังก์ชั่นและเอาท์พุท

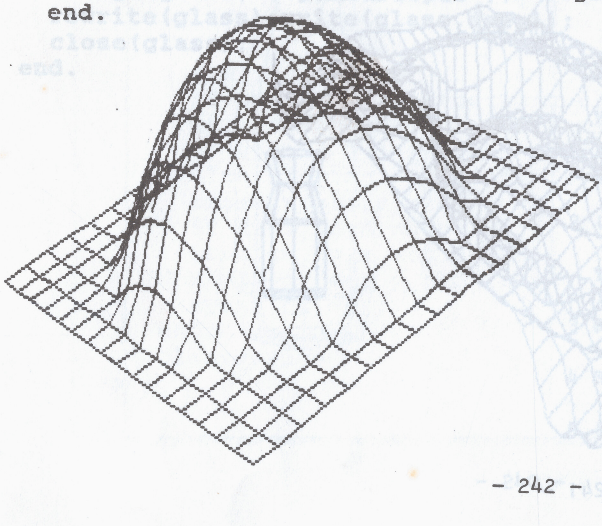
```
function y_value(x,z:real):real;
var r:real;
begin
  r:=(x*x/2+z*z/2);
  y_value:=2*sin(r*pi/3)+sin(r*2*pi/3+0.65);
end;
```



SIN.PAS

ตัวอย่างที่ 11.13 การเขียนกราฟฟังก์ชันที่มีตัวแปร 2 ตัวแปร

```
program graphics_of_function;
uses graph,crt;
type _2dpoint=record
              x:integer; y:integer
            end;
   _3dpoint=record
              x:real;y:real;z:real
            end;
var i,j,k,x_old,y_old:integer;
    x_sin,y_sin,z_sin,x_cos,y_cos,z_cos,x_shift,y_shift,z_shift:real;
    a,c,d,e:_3dpoint;  f:_2dpoint;
{$I b:graph4.pas }
   function y_value(x,z:real):real;                     ...................... 1
   var r,y:real;
     begin
       r:=(x*x/10+z*z/10); if (r>10)  then
       y_value:=0 else y_value:=(10-r);
     end;
begin
   a.x:=30;a.y:=30;a.z:=30; rotation(a);
   j:=detect; initgraph(j,k,'');
   x_shift:=0;y_shift:=0;z_shift:=0;
   for i:=1 to 2 do                                     ...................... 2
   begin c.z:=-12;                                      ...................... 3
     repeat c.x:=-12;                                   ...................... 4
       repeat
         c.y:=y_value(c.x,c.z);
         case i of
            1:trans(c,e);
            2:begin d.x:=c.z;d.z:=c.x;d.y:=c.y;trans(d,e) end;
         end;
         purse(e,f);
         if (c.x>-12) then line (x_old,y_old,f.x,f.y);  ......... 5
         x_old:=f.x; y_old:=f.y;c.x:=c.x+1.5;
       until (c.x>12) or keypressed;
       c.z:=c.z+1.5;
     until  (c.z>12) or keypressed;
   end;
   repeat until keypressed;  closegraph
end.
```



1 สร้างฟังก์ชัน y_value

2 ระหว่างที่ i เป็น 1 ให้ z คงที่และเปลี่ยนค่า x และสลับค่า x กับ y เมื่อ i เป็น 2

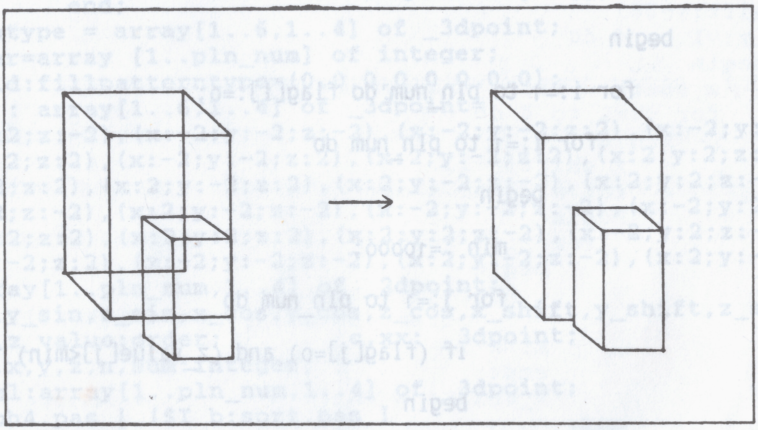
3 ให้ค่าเริ่มต้นกับ x

4 ให้ค่าเริ่มต้นกับ z

5 ลากเส้นระหว่างค่าที่แล้วกับค่าที่คำนวณได้

< การเขียนกราฟฟิกวัตถุซ้อนกัน >

การเขียนกราฟฟิกวัตถุซ้อนกัน ดังรูปข้างล่างมีวิธีเขียนหลายวิธีด้วยกัน สำหรับในหนังสือเล่มนี้ใช้วิธีเขียนจากด้านไกลก่อน แล้วจึงเขียนด้านใกล้ กล่าวคือ ทำการเรียงลำดับระยะทางจากตาถึงค่าเฉลี่ยของแต่ละด้าน ด้านใดที่มีค่า z น้อย หมายถึงอยู่ไกลให้เขียนด้านนั้นก่อน



ขั้นตอนการเขียนวัตถุซ้อน โดยวิธีการเรียงลำดับระยะทางมีดังนี้คือ

สมมุติให้ a b และ c เป็นด้าน 3 ด้าน

1 หาผลบวกของค่าแกน z ของด้านแต่ละด้าน (หรือหาค่าเฉลี่ย)

2 เรียงลำดับค่าที่คำนวณได้จากข้อ 1 จากน้อยไปมาก สมมุติว่าลำดับที่ได้คือ b a และ c

3 เขียนและระบายด้าน b a และ c

ใน Turbo Pascal มีคำสั่ง fillpoly ซึ่งเป็นคำสั่งระบายรูปหลายเหลี่ยมที่เขียนโดยคำสั่ง drawpoly และใช้คำสั่ง setfillpattern เมื่อกำหนดวิธีระบาย ก็สามารถลบด้านที่อยู่ข้างหลัง โดยการเขียนด้านที่อยู่ไกลและระบายพื้นที่ภายในด้วยคำสั่ง fillpoly

การใช้คำสั่ง setfillpattern สามารถทำได้ดังนี้คือ

ก่อนอื่นให้ build เป็นค่าคงที่แบบ fillpatterntype ซึ่งมีโครงสร้างเป็น

```
type fillpatterntype =array[1..8] of byte;
const build:fillpatterntype=[$0,$0,$0,$0,$0,$0,$0,$0];
begin

    setfillpattern(build,1);
```

การเรียงลำดับข้อมูลของด้านสามารถคำนวณโดยใช้โพรซิเจอร์ sort ที่เห็นข้างล่าง

```
procedure sort(z_value:order;var prior:order);
  var flag :array[1..pln_num] of integer;
      i,j,min,num:integer;
  begin
    for i:=1 to pln_num do flag[i]:=0;
      for i:=1 to pln_num do
        begin
          min :=10000;
          for j:=1 to pln_num do
            if (flag[j]=0) and (z_value[j]<min) then
            begin
              min:=z_value[j]; num:=j;
            end;
            prior[i]:=num; flag[num]:=1
        end
  end;
```

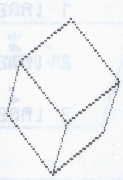
order เป็นแบบข้อมูลที่มีโครงสร้างดังที่เห็น

type order = array [1..จำนวนด้าน] of integer;

เมื่อโพรซิเจอร์ sort รับค่า z_value ก็จะดำเนินการ คำนวณว่า z_value ตัวที่เท่าไดมีค่าน้อยที่สุด (z_value เป็นอาร์เรย์เก็บผลบวกของจุดยอดทุกจุดของแต่ละด้าน) เมื่อได้ z_value ที่น้อยที่สุด (ด้านที่ไกลที่สุด) ก็ส่งอินเด็กซ์นั้นให้กับ num และให้ prior[1] เท่ากับ num ยกตัวอย่างเช่น สมมุติว่าด้านที่ไกลที่สุดคือด้านที่ 13 16.. 21 ผลที่ได้คือ prior[1] = 13 prior[2] = 16 .... prior[สุดท้าย] = 21 สำหรับตัวแปร flag เป็นตัวแปรที่มีค่าเป็น 0 สำหรับด้านที่ยังไม่ได้จัดลำดับ และเป็น 1 สำหรับด้านที่มีการจัดลำดับแล้ว

ตัวอย่างที่ 11.14 การวาดรูปกล่องสี่เหลี่ยม และทำการลบด้านที่อยู่ไกลออก

```
program box;
uses graph;
const pln_num = 6;
type _2dpoint=record
                 x:integer; y:integer
               end;
   _3dpoint=record
                 x:real;y:real;z:real
               end;
     datatype = array[1..6,1..4] of _3dpoint;
     order=array [1..pln_num] of integer;
const build:fillpatterntype=(0,0,0,0,0,0,0,0);
     pnt : array[1..6,1..4] of _3dpoint=
(((x:-2;y:2;z:-2),(x:-2;y:-2;z:-2),(x:-2;y:-2;z:2),(x:-2;y:2;z:2)),
 ((x:-2;y:2;z:2),(x:-2;y:-2;z:2),(x:2;y:-2;z:2),(x:2;y:2;z:2)),
 ((x:2;y:2;z:2),(x:2;y:-2;z:2),(x:2;y:-2;z:-2),(x:2;y:2;z:-2))        ...1
 ((x:2;y:2;z:-2),(x:2;y:-2;z:-2),(x:-2;y:-2;z:-2),(x:-2;y:2;z:-2)),
 ((x:-2;y:2;z:2),(x:2;y:2;z:2),(x:2;y:2;z:-2),(x:-2;y:2;z:-2)),
 ((x:-2;y:-2;z:2),(x:-2;y:-2;z:-2),(x:2;y:-2;z:-2),(x:2;y:-2;z:2)));
var xs:array[1..pln_num,1..4] of _2dpoint;
    x_sin,y_sin,z_sin,x_cos,y_cos,z_cos,x_shift,y_shift,z_shift:real;
    prior,z_value:order;          c,xx: _3dpoint;
    i,j,k,x,y,z,n,sum:integer;
    pntreal:array[1..pln_num,1..4] of _3dpoint;
{$i b:graph4.pas } {$I b:sort.pas }
  procedure makedata;
     begin
        x_shift:=0; y_shift:=0; z_shift:=0;
        for i:= 1 to pln_num do
           for j:=1 to 4 do
              trans(pnt[i,j],pntreal[i,j]);
     end;
begin  { main }
  c.x:=30;c.y:=20;c.z:=10;
  rotation(c);    makedata;   (* call data *)
  i:=detect; initgraph(i,j,'');
  for i:=1 to pln_num do   (* 6 *)
     begin sum:=0;
           for j:=1 to 4 do   (* 4 *)
              begin
                trans(pntreal[i,j],xx); purse(xx,xs[i,j]);
                sum:=sum+round(xx.z)
              end; z_value[i]:=sum;
     end;    sort(z_value,prior);                  ........................ 2
  for i:=1 to pln_num do
     begin
        n:=prior[i];
        setfillpattern(build,1); ..3
        fillpoly(4,xs[n]);      ...4
     end;
  readln; closegraph;
end.
```



1 ข้อมูลของแต่ละด้าน

2 ทำการเรียงลำดับจากด้านไกลไปใกล้

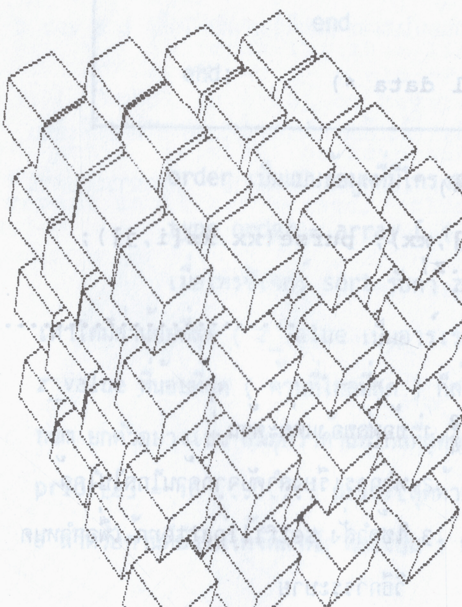
3 ใช้คำสั่ง setfillpattern เพื่อกำหนดวิธีการระบาย

4 ใช้คำสั่ง fillpoly ลบด้านที่อยู่ข้างหลัง

ตัวอย่างที่ 11.15 การวาดรูปกล่องสี่เหลี่ยม ซ้อนกัน 64 ใบ และทำการเขียนกราฟฟิก แบบซ้อนกัน

กำหนดให้ ตัวคงที่ pln_num = 384; และสร้าง โพรซิเจอร์ makedata ใหม่

```
procedure makedata;
begin
   k:=1;
   z_shift:=-35;
   for z:=1 to 4 do                                    { z axis }
      begin
         x_shift:=-14; y_shift:=-21;
         for y:=1 to 4 do                              { y axis }
            begin
               x_shift:=-13;
               for x:=1 to 4 do                        { x axis }
               begin
                  for i:=1 to 6 do                     { plane }
                     begin
                        for j:=1 to 4 do               { point }
                           trans(pnt[i,j],pntreal[k,j]);
                        k:=k+1;
                        end;
                        x_shift:=x_shift+6;         ........... 1
               end;
               y_shift:=y_shift+6;                  ........... 2
            end;
      z_shift:=z_shift+8;                           ........... 3
      end;
end;
```



1 เคลื่อนไปทางแกน z ทีละ 6

2 เคลื่อนไปทางแกน y ทีละ 6

3 เคลื่อนไปทางแกน x ทีละ 6

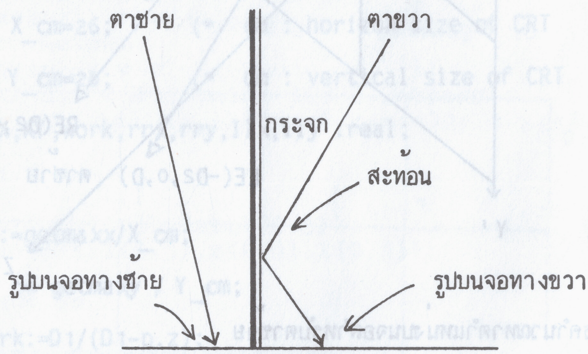
< สเตริโอกราฟฟิก (stereo graphics) >

การที่สายตาของมนุษย์สามารถเห็นวัตถุมีส่วนลึกหรือตื้นได้ เกิดจากการปรากฏของภาพตกลงบนสายตาทั้งสองต่างกัน (parallax) เมื่อนำเอาหลักการนี้ประยุกต์ใช้กับกราฟฟิก ก็จะเกิดภาพซึ่งสามารถมองเห็นความตื้นลึกได้

การแสดงสเตริโอกราฟฟิกมีหลายวิธีด้วยกัน แต่ในที่นี้จะกล่าวถึงวิธีที่ไม่ซับซ้อนนั่นคือ

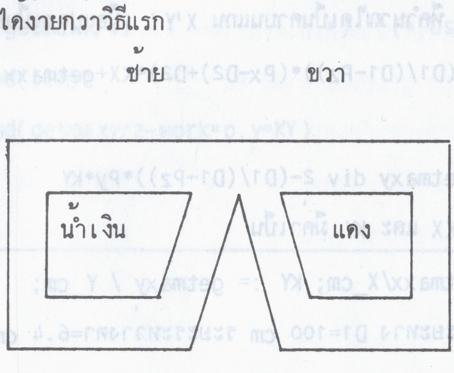
1 วิธีใช้กระจก

จากรูป การเขียนสเตริโอกราฟฟิกโดยใช้กระจกนั้น มีหลักการดังนี้คือ ตาของมนุษย์จะมองไปที่จุดๆเดียว ดังนั้น จึงเขียนกราฟฟิกของวัตถุขึ้น 2 รูป รูปสำหรับตาซ้ายให้อยู่ทางซ้ายส่วนรูปสำหรับตาขวาให้อยู่ข้างขวา ใช้กระจกคั่นตรงกลางและมองที่จอตามรูป ตาซ้ายจะเห็นรูปทางซ้ายส่วนตาขวาจะเห็นรูปที่สะท้อนจากกระจก

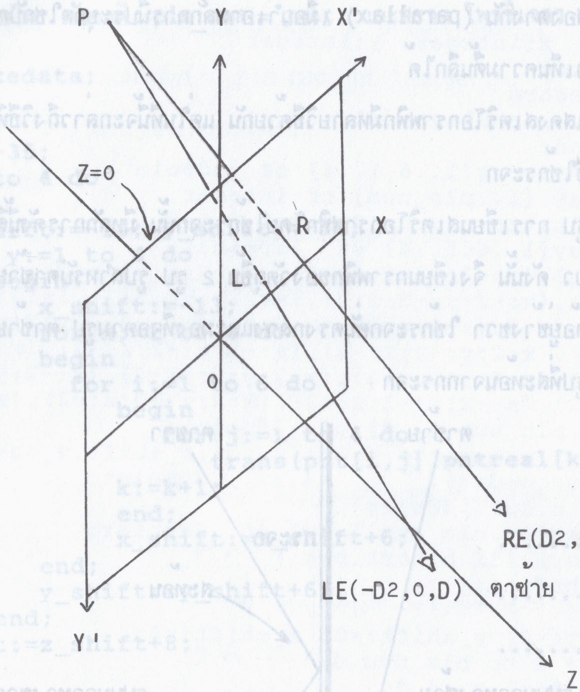


2 วิธีใช้แว่นตาสามมิติ

วิธีการใช้แว่นตาสามมิติมีหลักการง่ายๆ ดังนี้คือ ใส่แว่นตากระดาษสีน้ำเงินที่ตาซ้าย และสีแดงที่ตาขวา การเขียนกราฟฟิกจะเขียนเป็นเส้นสีน้ำเงินสำหรับตาซ้าย และสีแดงสำหรับตาขวา เนื่องจากแว่นตาสีน้ำเงินจะเห็นแต่เส้นสีน้ำเงิน และแว่นตาสีแดงจะเห็นแต่เส้นสีแดง วิธีนี้เป็นวิธีที่มองเห็นภาพสเตริโอสามมิติได้ง่ายกว่าวิธีแรก



< การคำนวณหาตำแหน่งที่ปรากฏบนจอสำหรับตาซ้ายและขวา >



การคำนวณหาตำแหน่งบนจอสำหรับตาซ้าย

จากสูตรที่ 1 และ 2 ของหัวข้อที่ 11.4 เมื่อแทนค่า (X1,Y1,Z1)และ(X2,Y2,Z2) ด้วย จุด (D2,0,D1) และ (Px,Py,PZ) ก็จะได้

x=(D1/(D1-pz))*(px-D2)+D2 ..... (1)

y=(D1/(D1-pz))*py

ค่า x,y ที่คำนวณได้เป็นค่าบนแกน X'Y เมื่อแปลงเป็นค่า Rx,Ry จะได้เป็น

Rx = ((D1/(D1-Pz))*(Px-D2)+D2)*KX+getmaxx div 2

..... (2)

Ry = getmaxy div 2-(D1/(D1-Pz))*Py*KY

เมื่อให้ KX และ KY มีค่าเป็น

KX:=getmaxx/X_cm; KY := getmaxy / Y_cm; ..... (3)

เมื่อให้ระยะทาง D1=100 cm ระยะระหว่างตา=6.4 cm D2 จะมีค่าเป็น 3.2 cm

ในทำนองเดียวกัน เมื่อคำนวณตำแหน่งวัตถุที่ปรากฏบนจอที่ตาซ้ายมองเห็นจะเท่ากับ

Lx = ((D1/(D1-Pz))*(Px+3.2)-3.2)*KX+getmaxx div 2 ..... (4)

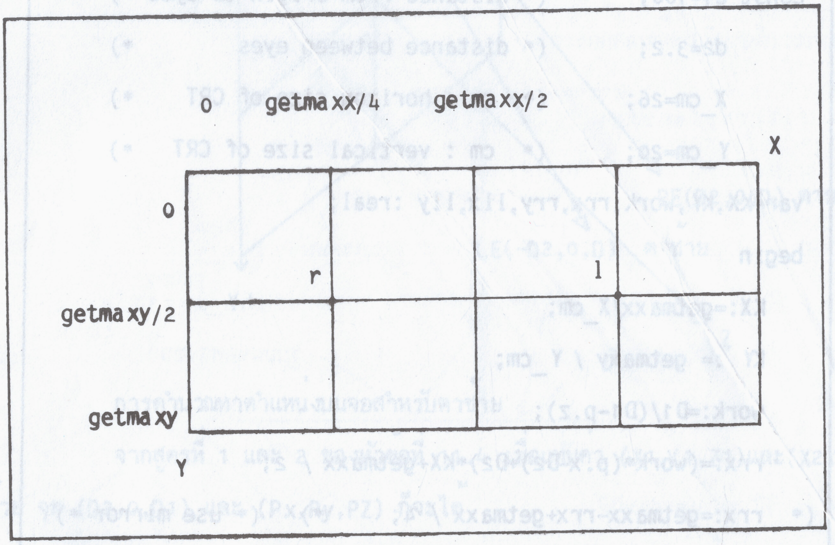
Ly = getmaxy div 2-(D1/(D1-Pz))*Py*KY

เมื่อเขียนเป็น โพรซิเจอร์จะได้เป็นดังที่เห็น

```
procedure stereo(p: 3dpoint;var r,l: 2dpoint);
const d1=100;        (* distance from origin to eyes *)
      d2=3.2;        (* distance between eyes        *)
      X_cm=26;       (* cm : horizon size of CRT     *)
      Y_cm=20;       (* cm : vertical size of CRT    *)
var KX,KY,work,rrx,rry,llx,lly :real;
begin
   KX:=getmaxx/X_cm;
   KY := getmaxy / Y_cm;
   work:=D1/(D1-p.z);
   rrx:=(work*(p.x-D2)+D2)*KX+getmaxx / 2;
(* rrx:=getmaxx-rrx+getmaxx / 4;         *)   (* use mirror *)
   r.x:=round(rrx);
   r.y:=round(getmaxy/2-work*p.y*KY);
   llx:=(work*(p.x+D2)-D2)*kX+getmaxx / 2;
(* llx:=llx-getmaxx/4;                   *)   (* use mirror *)
   l.x:=round(llx);
   l.y:=round(getmaxy/2-work*p.y*KY)
end;
```

จากโพรซิเจอร์ที่เห็น ค่า r และ l คือจุดที่มองเห็นบนจอโดยตาขวาและตาซ้าย ตามลำดับ ซึ่งจะเห็นได้ว่าโพรซิเจอร์นี้เป็นโพรซิเจอร์ที่เขียนรูปทับกัน ดังนั้น ถ้าหากใช้โพรซิเจอร์ข้างบน ก็ต้องลากเส้นที่มองด้วยตาขวาด้วยสีแดง (setcolor(4)) และ เส้นที่มองด้วยตาซ้ายด้วยสีน้ำเงิน (Setcolor(1)) และใช้แว่นตาสามมิติดูรูปกราฟฟิก

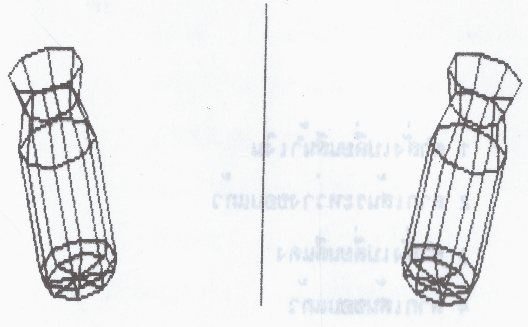
สำหรับมีข้อความที่อยู่ภายในเครื่องหมาย (* *) คือข้อความที่จำเป็นต้องใช้ในกรณีที่ใช้กระจกดูสเตริโอกราฟฟิก ซึ่งตำแหน่งและลักษณะของรูปสำหรับตาขวาและตาซ้ายที่ปรากฏบนจอจะเปลี่ยนตำแหน่งเป็นดังที่เห็น



สำหรับรูปที่จะนำมาใช้ในการวาดกราฟฟิก จะนำรูปที่เก็บในไฟล์ชื่อ picture.pic จากตัวอย่างที่ 11.12 ซึ่งเป็นโปรแกรมวาดรูปโดยผู้ใช้วาดรูปเอง และเก็บข้อมูลของรูปลงไฟล์ สำหรับโพรซิเจอร์ของการคำนวณจุดของสเตริโอกราฟฟิกแบบใช้กระจกและแบบใช้แว่นตาจะเก็บไว้ในไฟล์ชื่อ stereo และ glasses ตามลำดับ

ตัวอย่างที่ 11.16 สเตริโอกราฟฟิกโดยใช้กระจก

```
program stereo_graph1;
uses graph,crt;
type _2dpoint=record
              x:integer; y:integer
           end;
    _3dpoint=record
              x:real;y:real;z:real
           end;
    datatype= array[0..8,1..10] of _3dpoint;
var data,data1 :datatype;
    x_sin,y_sin,z_sin,x_cos,y_cos,z_cos,x_shift,y_shift,z_shift:real;
    c : _3dPoint;
    r,l,rr,ll:array[0..8,1..100] of _2dpoint;
    glasstype:file of datatype;
    graphdriver,graphmode,k,j:integer;
{$I b:graph4.pas }  {$I b:stereo.pas }
begin
   assign(glasstype,'b:picture.pic');                        ............ 1
   reset(glasstype); read(glasstype,data);
   graphdriver:=detect;
   initgraph(graphdriver,graphmode,'a:\');
   line(getmaxx div 2,0,getmaxx div 2,getmaxy);
   x_shift:=0;y_shift:=0;z_shift:=0;c.x:=30; c.y:=30; c.z:=30;
   rotation(c);
   for j:=1 to 6 do
      begin
         trans(data[0,j],data1[0,j]);
         stereo(data1[0,j],r[0,j],l[0,j]);                    ........... 2
      end;
   for k:=1 to 8 do
      begin
         for j:=1 to 6 do
            begin
               trans(data[k,j],data1[k,j]);
               stereo(data1[k,j],r[k,j],l[k,j]);
               line(r[k-1,j].x,r[k-1,j].y,r[k,j].x,r[k,j].y);
               line(l[k-1,j].x,l[k-1,j].y,l[k,j].x,l[k,j].y); ...... 3
            end;
      drawpoly(6,r[k]); drawpoly(6,l[k])                     .............. 4
  end;
repeat until keypressed;
closegraph; close(glasstype)
end.
```



1 เรียกไฟล์ picture.pic

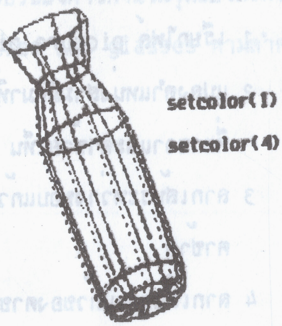
2 แปลงตำแหน่งสามมิติมาที่ตำแหน่งบนจอที่ตาขวาและตาซ้ายเห็น

3 ลากเส้นระหว่างขอบแก้วของตาขวาและตาซ้าย

4 ลากเส้นขอบแก้วของตาขวาและตาซ้าย

ตัวอย่างที่ 11.17 สเตริโอกราฟฟิกโดยใช้แว่นตาสามมิติ

```
program stereo_graph2;
uses graph,crt;
type _2dpoint=record
          x:integer; y:integer
        end;
     _3dpoint=record
          x:real;y:real;z:real
        end;
     datatype= array[0..8,1..10] of _3dpoint;
var  data,data1 :datatype;
     x_sin,y_sin,z_sin,x_cos,y_cos,z_cos,x_shift,y_shift,z_shift:real;
     c : _3dPoint;
     r,l,rr,ll:array[0..8,1..100] of _2dpoint;
     glasstype:file of datatype;
     graphdriver,graphmode,k,j:integer;
{$I b:graph4.pas }  {$I b:glasses.pas }
begin
   assign(glasstype,'b:picture.pic');
   reset(glasstype); read(glasstype,data);
   graphdriver:=detect;
   initgraph(graphdriver,graphmode,'a:\');
   x_shift:=0;y_shift:=0;z_shift:=0;c.x:=20; c.y:=30; c.z:=30;
   rotation(c);
   for j:=1 to 6 do
      begin
         trans(data[0,j],data1[0,j]);
         stereo(data1[0,j],r[0,j],l[0,j]);
      end;
   for k:=1 to 8 do
      begin
         for j:=1 to 6 do
            begin
               trans(data[k,j],data1[k,j]);
               stereo(data1[k,j],r[k,j],l[k,j]);
               setcolor(1);                                          ..........1
               line(r[k-1,j].x,r[k-1,j].y,r[k,j].x,r[k,j].y);       .... 2
               setcolor(4);                                          .... 3
               line(l[k-1,j].x,l[k-1,j].y,l[k,j].x,l[k,j].y);
            end;
      drawpoly(6,l[k]); setcolor(1); drawpoly(6,r[k])           ......... 4
  end;
repeat until keypressed;
closegraph; close(glasstype)
end.
```



1 คำสั่งเปลี่ยนสีน้ำเงิน

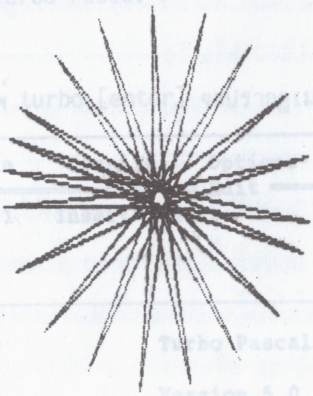
2 ลากเส้นระหว่างขอบแก้ว

3 คำสั่งเปลี่ยนสีแดง

4 ลากเส้นขอบแก้ว

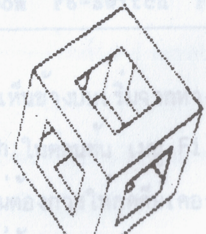
แบบฝึกหัด

1 จงใช้ turtle graphics วาดรูปที่เห็นข้างล่าง



2 จากรูปที่เขียนได้จากตัวอย่างที่ 11.11 จงทำการหมุนรอบแกนทั้งสามและแสดงออกจอ

3 จงเขียนกราฟฟิกสามมิติดังที่เห็นข้างล่าง



```
program stereo_graph2;
uses graph,crt;
type _2dpoint=record
                x:integer; y:integer
              end;
     _3dpoint=record
                x:real;y:real;z:real
              end;
     datatype= array[0..8,1..10] of _3dpoint;
var  data,data1 :datatype;
     x_sin,y_sin,z_sin,x_cos,y_cos,z_cos,x_shift,y_shift,z_shift:rea
     c : _3dPoint;
     r,l,rr,ll:array[0..8,1..100] of _2dpoint;
     glasstype:file of datatype;
     graphdriver,graphmode,k,j:integer;
{$I b:graph4.pas }  {$I b:glasses.pas }
begin
   assign(glasstype,'b:picture.pic');
   reset(glasstype); read(glasstype,data);
   graphdriver:=detect;
   initgraph(graphdriver,graphmode,'a:\');
   x_shift:=0;y_shift:=0;z_shift:=0;c.x:=20; c.y:=30; c.z:=30;
   rotation(c);
   for j:=1 to 6 do
     begin
       trans(data[0,j],data1[0,j]);
       stereo(data1[0,j],r[0,j],l[0,j]);
     end;
   for k:=1 to 8 do
     begin
       for j:=1 to 6 do
         begin
           trans(data[k,j],data1[k,j]);
           stereo(data1[k,j],r[k,j],l[k,j]);
           setcolor(1);
           line(r[k-1,j].x,r[k-1,j].y,r[k,j].x,r[k,j].y);
           setcolor(4);
           line(l[k-1,j].x,l[k-1,j].y,l[k,j].x,l[k,j].y);
         end;
     drawpoly(6,l[k]); setcolor(1); drawpoly(6,r[k])
  end;
repeat until keypressed;
closegraph; close(glasstype)
end.
```

ภาคผนวก ก

< การใช้เมนูของ Turbo Pascal >

โดยปรกติเมื่อพิมพ์ turbo [enter] จะปรากฏเมนูขึ้นที่จอดังข้างล่าง

```
 File     Edit     Run     Compile   Options   Debug     Break/watch
 ════════════════════════════ Edit ═══════════════════════════════════
     Line 1    Col 1    Insert  Indent          Unindent   A:NONAME.PAS

                      ┌────────────────────────────────┐
                      │          Turbo Pascal          │
                      │          Version 5.0           │
                      │   Copyright (c) 1983, 1988 by  │
                      │  Borland International, Inc.   │
                      └────────────────────────────────┘

 ──────────────────────────────── Watch ──────────────────────────────

 F1-Help  F5-Zoom  F6-Switch  F7-Trace  F8-Step  F9-Make  F10-Menu
```

เมนูที่เห็นข้างบนเริ่มจากทางขวาสุดคือ File Edit Compile Options Debug และ Break/Watch ในตอนต้น เมนู File จะปรากฏสีเข้มกว่าเมนูอื่น ถ้าต้องการใช้เมนู File ให้กด enter ถ้าไม่ต้องการให้กดคีย์เคอร์เซอร์ (←↑↓→) เพื่อเลือกเมนูอื่นและกด enter เพื่อใช้งาน และในกรณีที่ต้องการออกจากเมนูใด ให้กดปุ่ม Esc เมนูทั้ง 7 มีหน้าที่ดังนี้คือ

File : จัดการเกี่ยวกับการอ่านหรือบันทึกโปรแกรม

Edit : เขียนหรือแก้ไขโปรแกรม

Run : ดำเนินการโปรแกรม

Compile : คอมไพล์โปรแกรม

Options : เลือกลักษณะการคอมไพล์และการทำงานอื่นๆ

Debug : ตรวจสอบหรือแก้ไขตัวแปรที่ต้องการ

Break/Watch : แก้ไขตัวแปรหรือนิพจน์ที่จอ Watch และแก้ไข Breakpoint

< เมนู File >

```
File
-----------------
Load        F3
Pick    Alt-F3
New
Save        F2
Write to
Directory
Change dir
OS shell
Quit     Alt-X
```

เมนู File เป็นเมนูเริ่มต้นในการเข้าสู่ Turbo Pascal โดยมีเมนูย่อยดังที่เห็น การทำงานของเมนูย่อยต่างๆมีดังนี้

1) Load หรือ F3

เมื่อเลือกเมนูนี้ (โดยกด enter) และใส่ชื่อไฟล์ที่ต้องการ ไฟล์ดังกล่าวจะปรากฏบนจอ

ในกรณีที่ไม่มีไฟล์ดังกล่าวจะเป็นไฟล์ใหม่

นอกจากนี้ยังสามารถกด enter โดยไม่ต้องใส่ชื่อไฟล์ใดๆ เพื่อดูไฟล์ที่มีทั้งหมดและเลื่อนคีย์เคอร์เซอร์เพื่อเลือกไฟล์ที่ต้องการ

2) Pick หรือ F3

เมนูนี้สามารถเลือก 8 ไฟล์สุดท้ายที่ได้ทำอ่านจากไฟล์หรือบันทึกลงไฟล์

3) New

เมนูสำหรับเปิดไฟล์เพื่อเขียนโปรแกรมใหม่ โดยถูกกำหนดชื่อเป็น NONAME.PAS ซึ่งสามารถเปลี่ยนเป็นชื่ออื่นก่อนบันทึกลงดิสก์

4) Save หรือ F2

เมนูสำหรับบันทึกโปรแกรมปัจจุบัน (ที่อยู่บน Editor) ลงดิสก์

5) Write to

เมนูสำหรับบันทึกโปรแกรมปัจจุบันลงดิสก์ โดยสามารถตั้งชื่อได้ตามที่ต้องการ

6) Directory

แสดงไฟล์ที่อยู่ในไดเร็กทอรีปัจจุบัน และสามารถอ่านไฟล์ที่ต้องการได้เช่นกัน

7) Change dir

เปลี่ยนไดเร็กทอรีปัจจุบันเป็นไดเร็กทอรีที่ต้องการ

8) OS shell

ออกจาก Turbo Pascal เข้าสู่ DOS เมื่อต้องการกลับเข้าสู่ Turbo Pascal ให้พิมพ์ exit และกด enter

9) Quit หรือ Alt-X

ออกจาก Turbo Pascal เข้าสู่ DOS โดยไม่สามารถกลับเข้าสู่ Turbo Pascal

< เมนู Edit >

Edit

เมื่อเลือกเมนูนี้ คีย์เคอร์เซอร์จะปรากฏบนจอส่วนล่างเพื่อเตรียมพร้อมที่จะให้ผู้ใช้เขียนหรือแก้ไขโปรแกรม รายละเอียดของ Editor ที่ปรากฏบนจอล่างเมนูหลักมีดังที่เห็น

```
Line 1    Col 1    Insert Indent         U nindent    A:NONAME.PAS
```

Line N　บอกบรรทัดที่เคอร์เซอร์อยู่

Col N　บอกคอลัมน์ที่เคอร์เซอร์อยู่

Insert　สามารถเขียนแทรกข้อความได้ (ถ้าต้องการให้เขียนทับกด Ctrl-v)

Indent　เมื่อขึ้นบรรทัดใหม่ จะย่อหน้าให้ตรงกับบรรทัดบน (กด Ctrl-OI ถ้าต้องการยกเลิก)

Tab　เมื่อกด Tab เคอร์เซอร์จะย้ายไปตรงกับย่อหน้า (กด Ctrl-OT ถ้าต้องการยกเลิก)

Unindent　เมื่อกด Backspace เคอร์เซอร์ถอยกลับตามตำแหน่งของย่อหน้า (กด Ctrl OU ถ้าต้องการยกเลิก)

สำหรับส่วนล่างของจอจะปรากฏ

```
F1-help  F5-Zoom  F6-Switch  F7-Trace  F8-Step  F9-Make  F10-Menu
```

รายละเอียดมีดังนี้คือ

F1　เปิดจอแสดงรายละเอียดการใช้งาน Turbo Pascal

F5　ขยายจอที่เคอเซอร์อยู่ให้เต็มจอ

F6　สลับระหว่างจอ Editor กับจอ Watch

F7　ดำเนินการโปรแกรมทีละหนึ่งข้อความ

F8　ดำเนินการโปรแกรมทีละหนึ่งข้อความ โดยจะดำเนินการข้ามโพรซิเจอร์

F9　คอมไพล์โปรแกรมหรือยูนิต

F10　ออกจากเมนู Editor ไปที่เมนู

< เมนู Run >

```
Run
Run              Ctrl-F9
Program reset    Ctrl-F2
Go to cursor          F4
Trace into            F7
Step over             F8
User screen       Alt-F5
```

เมนูนี้ทำหน้าที่เลือกการดำเนินการ ซึ่งสามารถดำเนินได้หลายวิธีดังนี้คือ

1) Run หรือ Ctrl-F9

เมื่อเลือกเมนูนี้ จะดำเนินการโปรแกรมแบบปรกติ

2) Program reset หรือ Ctrl-F2

หน้าที่ของเมนูนี้คือ ยกเลิกดำเนินการคำสั่งต่างๆที่ใช้ในการตรวจโปรแกรมของเมนู Debug และเลิกการจองหน่วยความจำของโปรแกรมนี้ และปิดไฟล์ที่เปิดทิ้งไว้

3) Go to Cursor หรือ F4

ดำเนินการโปรแกรมจนกระทั่งถึงตำแหน่งที่เคอร์เซอร์อยู่ ในกรณีที่เคอร์เซอร์อยู่ที่ตำแหน่งที่ไม่สามารถดำเนินการได้ จะดำเนินการเลยไปจนถึงตำแหน่งที่สามารถดำเนินการได้

4) Trace into หรือ F7

ดำเนินการโปรแกรมทีละข้อความ คำสั่งนี้จะดำเนินการข้อความทุกข้อความ ไม่ว่าจะเป็นข้อความในโพรซิเจอร์หรือในไฟล์ที่เรียกจากนอกโปรแกรม

5) Step over หรือ F8

ดำเนินการโปรแกรมทีละข้อความเช่นเดียวกับคำสั่ง Trace into สิ่งที่ต่างกัน คือจะดำเนินการข้ามโพรซิเจอร์หรือไฟล์ที่มีการเรียกใช้

6) User screen หรือ Alt-F5

แสดงจอของเอาท์พุท กล่าวคือ เปลี่ยนการแสดงจาก Editor มาเป็นแสดงเอาท์พุท การดำเนินการแสดงเอาท์พุทนั้นจะแสดงในช่วงเวลาสั้นๆ คำสั่งนี้จะทำให้เห็นเอาท์พุทได้นานตามตามต้องการ เมื่อต้องการกลับไปจอเดิม ก็ใช้คำสั่งเดียวกัน

<เมนู Compile>

```
Compile
Compile Alt-F9
Make F9
Build
Destination Memory
Find error
Primary file:
Get info
```

เมนูนี้สำหรับเลือกแบบการคอมไพล์ ซึ่งรายละเอียดมีดังนี้

1) Compile หรือ Alt-F9

คำสั่งนี้จะคอมไพล์โปรแกรมหรือยูนิตตามปรกติ

2) Make หรือ F9

คำสั่งนี้จะคอมไพล์โปรแกรมหรือยูนิต ที่ได้กำหนดใน Primary file ในกรณีที่ไม่มีไฟล์ใดๆที่ Primary file จะคอมไพล์โปรแกรมหรือยูนิตที่อยู่บน Editor ในกรณีที่มีชื่อไฟล์ใน Primary file เมื่อใช้คำสั่ง make จะมีการคอมไพล์โปรแกรมหรือยูนิตใน Primary file ซึ่งโดยปรกติแล้ว จะมีการเรียกใช้โปรแกรมหรือยูนิตที่อยู่ใน Editor ปัจจุบัน คำสั่ง make จะดำเนินการคอมไพล์ไฟล์เป็นไฟล์ .tpu ใหม่ ในกรณีที่

1 มีการเปลี่ยนแปลงโปรแกรมใน primary file

2 มีการเปลี่ยนแปลงไฟล์หรือยูนิตที่มีการเรียกจาก Primary file

3) bulid

คำสั่งนี้ดำเนินการคล้ายคลึงกับคำสั่ง make สิ่งที่ต่างคือ คำสั่งนี้จะดำเนินการสร้างไฟล์ .tpu ใหม่ทุกกรณี

4) destination

คำสั่งนี้สามารถกำหนดให้คอมไพล์ไปที่หน่วยความจำหรือดิสก์ ซึ่งเลือกโดย Space Bar

5) Fine error

เมื่อมี error หลังการดำเนินการ เมนูนี้จะแสดงตำแหน่งหน่วยความจำที่เกิด error โดยจะแสดงเป็นลักษณะ เซกเมนท์ : ออฟเซท

6) Primary file :

เมนูนี้เป็นเมนูสำหรับโปรแกรมที่มีการเรียกโปรแกรมหรือยูนิตอื่นๆ เมื่อใส่โปรแกรมหลักไว้ที่เมนูนี้ และทำการแก้ไขโปรแกรมหรือยูนิตบน Editor การคอมไพล์โดยใช้คำสั่งต่างๆข้างบนจะสะดวกมากขึ้นเนื่องจากไม่ต้องทำการเรียกไฟล์หลายๆครั้ง

7) Get info

แสดงรายละเอียดต่างๆเช่น ชื่อไฟล์ ขนาดแสตก และขนาดฮีพ เป็นต้น

< เมนู Options >

Options

- Compiler
- Linker
- Environment
- Directories
- Parameters
- Save options
- Retrieve options

เมนูย่อยของเมนู Options มีดังนี้คือ

1) Compiler

เมนูนี้จะแสดงการสั่งงานคอมไพเลอร์ต่างๆ ซึ่งอยู่ในภาคผนวก ค สำหรับ Conditional Defines คือ เมนูกำหนดสัญลักษณ์เงื่อนไข และ Memory Sizes คือ เมนูกำหนดขนาดหน่วยความจำ

2) Linker

เมนูนี้มีเมนูย่อยอยู่ 2 เมนูคือ

1 Map File ซึ่งมีรายละเอียดดังนี้

off : ไม่สร้าง Map file

Segments : สร้าง Map file เก็บข้อมูลเซกเมนต์

Publics : สร้าง Map file เก็บชื่อ ตำแหน่งหน่วยความจำและตำแหน่งโปรแกรม

Detailed : ให้ข้อมูลเหมือน Publics และข้อมูลเลขบรรทัด และ module tables

2 Linker เมนูนี้ทำหน้าที่กำหนดแบบการเรียกระหว่างไฟล์ .obj ว่าต้องการให้เรียกโดยผ่านทางหน่วยความจำหรือใช้ดิสก์

3) Environment

รายละเอียดเมนู Environment มีดังต่อไปนี้

1 Config Auto Save เมนูนี้เมื่อเป็น on จะเก็บเงื่อนไขต่างๆเช่น การสั่งงานของคอมไพเลอร์ที่ได้กำหนดไว้ในเมนูลงไฟล์ turbo.tpc หรือไฟล์ .tp (ค่าปรกติเป็น off)

2 Edit Auto Save เมนูนี้เมื่อเป็น on จะบันทึกโปรแกรมใน Editor ลงดิสก์ทุกครั้งที่มีการดำเนินการโดยคำสั่ง Run หรือ Step Over (ค่าปรกติเป็น off)

3 Backup files เมนูนี้เมื่อเป็น on จะมีการเก็บไฟล์ .bak (ค่าปรกติเป็น on)

4 Tab size เมนูกำหนดขนาด tab (ค่าปรกติเป็น 8)

5 Screen size เมนูกำหนดบรรทัดต่อจอหนึ่งจอว่าต้องการอักษร 25 บรรทัด หรือ 43/50 บรรทัดต่อจอหนึ่งจอ

4) Directories

เมนูสำหรับกำหนดไดเร็กทอรีสำหรับไฟล์ต่างๆ ซึ่งมีรายละเอียดดังนี้

1 Turbo Directory เมนูนี้เป็นเมนูกำหนดไดเร็กทอรีสำหรับไฟล์ Turbo.hLp ซึ่งเป็นไฟล์บอกข้อมูลที่ผู้ใช้ต้องการโดยกด (F1) และ turbo.tp ซึ่งเป็นไฟล์เก็บรายละเอียดเค็บเงื่อนไขต่างๆ วิธีการกำหนดกำหนดอาจกำหนดเป็น A:

2 EXE & TPU Directory เป็นเมนูกำหนดไดเร็กทอรีสำหรับไฟล์ .EXE และ TPU

3 Include directories เป็นเมนูกำหนดไดเร็กทอรีสำหรับไฟล์ที่ต้องการจะเรียกมาใช้

4 Unit directories เป็นเมนูกำหนดไดเร็กทอรีสำหรับไฟล์ยูนิต

5 Object directories เป็นเมนูกำหนดไดเร็กทอรีสำหรับไฟล์ .obj

6 Pick file name เป็นเมนูสำหรับใส่ชื่อไฟล์ turbo.tpc หรือ ไฟล์ .tpc เพื่อเก็บไฟล์ 8 ไฟล์สุดท้ายและเรียกเข้าหน่วยความจำทุกครั้งที่มีการเรียก Turbo Pascal

7 Current pick file แสดงชื่อของไฟล์ที่เก็บโปรแกรมที่เรียกเข้ามา

5) Parameters ใส่ชื่อพารามิเตอร์ที่ต้องการส่ง command line (เป็นข้อมูลแบบสตริง)

6) Save Options บันทึกเงื่อนไขต่างๆที่ใช้ในการคอมไพล์และ ไฟล์ชุดต้องการจะเรียกโดยปรกติจะมีชื่อ turbo.tp เมื่อการใช้ Turbo Pascal จะเรียก turbo.tp โดยอัตโนมัติ เมื่อใช้ชื่ออื่นก็สามารถเรียกเงื่อนไขในการคอมไพล์และไฟล์ชุดอื่นที่ต้องการจะใช้ควบคู่กันไป(สมมุติว่าใช้ชื่อ word.tp และที่เมนู Pick file name มีชื่อ word.tpc และทำการบันทึกก็จะได้ไฟล์ .tp อีกไฟล์หนึ่ง)

7) Retrieve Options เรียกไฟล์.tp ที่เก็บเงื่อนไขการคอมไพล์และชุดไฟล์เข้ามา ในการใช้ Turbo Pascal นั้น โดยปรกติจะใช้วิธี

A>turbo [enter]

นอกจากการเรียกโดยวิธีนี้แล้วยังสามารถใช้วิธี

A>turbo /Cชื่อไฟล์.tp (ระหว่าง C กับชื่อไฟล์ต้องไม่มีเว้นวรรค)

เช่น A>turbo /Cword ดังตัวอย่างในข้อ 6) (Save Options) เมื่อเรียกโดยใช้คำสั่งที่เห็น จะเป็นการเรียกไฟล์ word.tpc และเงื่อนไขแบบ word.tp วิธีการเรียกอื่นมีดังนี้

A>turbo /ชื่อไฟล์.tp /B (ดำเนินการคอมไพล์แบบ build)

A>turbo /ชื่อไฟล์.tp /M (ดำเนินการคอมไพล์แบบ make)

A>turbo /D (เรียก Turbo Pascal โดยใช้จอ 2 จอในเวลาเดียวกัน)

< เมนู Debug >

Debug	
Evaluate	Ctrl F4
Call stack	Ctrl F3
Find procedure	
Integrated debugging	On
Standalone debugging	Off
Display swapping	Smart
Refresh display	

เมนูนี้มีสำหรับตรวจค่าต่างๆ

1) Evaluate หรือ Ctrl F4

เมนูนี้ควรใช้ควบคู่กับคำสั่ง Trace into (F7) หรือ Step over (F8) เมื่อเลือกเมนูนี้จะปรากฏช่อง 3 ช่อง ช่องแรก สำหรับใส่ตัวแปร (แบบใดๆก็ได้)ที่ต้องการทราบค่า เมื่อกด F7,F8 ค่าตัวแปรจะปรากฏที่ช่องที่ 2 ส่วนช่องสุดท้ายเป็นการให้ค่าตัวแปร นอกจากนี้ ช่องบนสุดยังสามารถเขียนนิพจน์ต่างๆเช่น 3*23.17 ซึ่งผลจะปรากฏที่ช่องที่ 2

2) Call stack หรือ Ctrl F3

เมนูนี้เมื่อใช้ควบคู่กับเมนู trace into จะทำให้ทราบถึงการเรียกจากโพรซิเจอร์หนึ่งไปอีกโพรซิเจอร์หนึ่งและอาจเรียกต่อไปกี่ชั้นก็ได้ หน้าต่างที่อยู่ทางขวาบนของจอแสดงการเรียกโพรซิเจอร์โดยเป็นลักษณะเหมือนข้อมูลแบบสแตก (last in first out)

3) Find Procedure

เมนูนี้เมื่อใส่ชื่อโพรซิเจอร์ใด เคอร์เซอร์จะปรากฏที่ตำแหน่งโพรซิเจอร์นั้นอยู่

4) Integrated debugging (ค่าปรกติ On)

เป็นเมนูการสั่งงานคอมไพเลอร์ Options/Compiler/Debug (ดูภาคผนวก ค)

5) Standalone debugging (ค่าปรกติ off)

เมื่อเมนูเป็น On และเมนู Compile/Destination เป็น Disk ไฟล์ .exe ที่ได้คอมไพล์ จะสามารถ ใช้ Turbo Debugger ตรวจหรือแก้ไขที่ผิดได้

6) Display Swapping (ค่าปรกติ Smart) รายละเอียดมีดังนี้

None เมื่อใช้คำสั่ง F8 จะไม่มีการสลับหน้าต่าง Editor กับหน้าต่างเอาท์พุท

Smart สลับหน้าต่างเฉพาะข้อความที่เป็นโพรซิเจอร์ (ทั้งที่เป็นและไม่เป็นแบบมาตรฐาน)

Always สลับหน้าต่างทุกข้อความ

7) Refresh display

เป็นเมนูแสดงจอเมนูหลักเดิม (ในกรณีที่มีการแสดงเอาท์พุททับจอเมนูหลัก)

< เมนู Break/watch >

```
Break/watch

Add watch                Ctrl F7
Delete watch
Edit watch
Remove all watches

Toggle breakpoint    Ctrl F8
Clear all breakpoints
View next breakpoint
```

รายละเอียดมีดังนี้

1) Add watch หรือ Ctrl F_7

เมนูนี้เป็นเมนูที่แสดงค่าตัวแปร นิพจน์ต่างๆ ตัวแปรไม่จำเป็นต้องเป็นตัวแปรประเภทพื้นฐาน กล่าวคือจะเป็นตัวแปรแบบอาร์เรย์หรือเรคอร์ดก็ได้ เมื่อใส่ชื่อตัวแปรและกดคีย์ F_7 หรือ F_8 จะเห็นการเปลี่ยนแปลงอย่างเห็นได้ชัดที่หน้าต่าง Watch เมื่อมีการใช้คำสั่งนี้เพื่อตรวจตัวแปรอื่นที่หน้าต่าง Watch จะปรากฏให้เห็นตัวแปรเก่าและใหม่ โดยตัวแปรหรือนิพจน์สุดท้ายจะมี (.) อยู่หน้าตัวแปรหรือนิพจน์นั้น

2) Delete watch

เมนูลบตัวแปรสุดท้ายที่หน้าต่าง Watch (หรือใช้วิธีเลื่อนเคอร์เซอร์ไปที่หน้าต่าง Watch และกด Del ก็ได้)

3) Edit watch

เมนูนี้มีสำหรับแก้ไขตัวแปรหรือนิพจน์สุดท้ายที่ต้องการตรวจทางหน้าต่าง Watch

4) Remove all watch

ลบตัวแปรหรือนิพจน์อยู่ในหน้าต่าง Watch ออกหมด

5) Toggle breakpoint หรือ Ctrl F_8

เมื่อเลื่อนเคอร์เซอร์มาที่ตำแหน่งที่ต้องการให้หยุดการดำเนินการและใช้คำสั่งนี้ (กดคีย์ Ctrl F_8) ข้อความที่บรรทัดนั้นจะปรากฏสีเข้ม และการดำเนินการจะหยุดที่ตำแหน่งดังกล่าว คำสั่งนี้อาจกำหนดหลายตำแหน่งก็ได้ โดยปรกติเมื่อดำเนินการมาถึงตำแหน่งนี้ ควรใช้คีย์ F_7, F_8 เพื่อตรวจโปรแกรม เมื่อต้องการยกเลิก ให้เลื่อนเคอร์เซอร์มาที่ตำแหน่งดังกล่าวและใช้คำสั่งเดิม

6) Clear all breakpoints

ลบตำแหน่งที่ต้องการให้หยุดการดำเนินการทุกตำแหน่ง

7) View next breakpoint

เลื่อนเคอร์เซอร์ไปที่ตำแหน่งที่ต้องการให้หยุดการดำเนินการถัดไป

ภาคผนวก ข

ฟังก์ชันและโพรซิเจอร์มาตรฐานใน Turbo Pascal

ฟังก์ชัน	แบบ	การดำเนินการ	พารามิเตอร์
addr(x)	pointer	ส่งแอดเดรสของ x	x:anytype
cseg	word	ส่งค่ารีจิสเตอร์ cs	none
dseg	word	ส่งค่ารีจิสเตอร์ ds	none
hi(i)	byte	ส่งค่าไบท์แรกของ i	i:integer or word
lo(i)	byte	ส่งค่าไบท์หลังของ i	
ofs(x)	word	ส่งค่าออฟเซ็ตของ x	x:anytype
paramcount	word	ส่งจำนวนพารามิเตอร์ที่ส่งมาทางเมนู command line	none
paramstr(x)	string	ส่งค่าพารามิเตอร์ทางเมนู	x:word
pi	real	ส่งค่า π	none
ptr(x,y)	pointer	ส่งค่าตัวแปรแบบพอยน์เตอร์ที่มีเซ็กเมนต์ที่ x และออฟเซ็ตที่ y	x:word y:word
seekeof(f)	bollean	ส่ง true ถ้าไฟล์พอยน์เตอร์ชี้ท้ายไฟล์	f:text
seekelon(f)	bollean	ส่ง true ถ้าชี้ที่ปลายบรรทัด	f:text
seg(x)	word	ส่งค่าเซ็กเมนต์ของ x	x:anytype
sizeof(x)	word	ส่งขนาดของตัวแปร x เป็นไบท์	x:anytype
sptr	word	ส่งค่ารีจิสเตอร์ sp	none
sseg	word	ส่งค่ารีจิสเตอร์ ss	none
swap(x)	same as parameter	ส่งค่าสลับไบท์สูงต่ำใน x	x:integer or word
upcase(c)	char	ส่งตัวอักษรเป็นตัวใหญ่	c:char
memavail	integer	ส่งขนาดทั้งหมดใน heap	none

โพรซิเจอร์	การดำเนินการ	พารามิเตอร์
append(f)	เปิดไฟล์เพื่อเพิ่มเติมข้อมูลที่ตำแหน่งสุดท้ายของไฟล์	f:text
blockread(f,buf,count [,result])	อ่านข้อมูล buf จำนวน count โดยไฟล์พอยน์เตอร์ f result เป็นจำนวนข้อมูลที่อ่านได้	f:untyped file buf:anytype count:word result:word
blockwrite(f,buf,count [,result])	บันทึกข้อมูล buf จำนวน count โดยไฟล์พอยน์เตอร์ f result เป็นจำนวนข้อมูลที่บันทึกได้	f:untyped file buf:anytype count:word result:word
chdir(s)	เปลี่ยนไดเร็กทอรีไปที่ s	s:string
dec(x[,n])	ลดตัวแปร x ลงจำนวน n หรือ ลดลง 1 เมื่อไม่มี n	x:scalar n:integer
exit	ออกจากบล็อกนั้นๆ	none
fillchar(x,count,c)	ให้ค่า c จำนวน count แก่ x	x:anytype count:word c:char
flush(f)	นำข้อมูลในบัฟเฟอร์เก็บลงไฟล์	f:text
getdir(d,s)	ส่งไดเร็กทอรีปัจจุบัน d ให้ s (0:ไดรฟ์ปัจจุบัน 1:a 2:b)	d:word s:string
halt([exitcode])	หยุดโปรแกรมที่กำลังดำเนินการอยู่และกลับสู่ dos	exitcode:word
inc(x[,n])	เพิ่มค่าตัวแปร x ขึ้นจำนวน n หรือเพิ่มขึ้น 1 เมื่อไม่มี n	x:scalar n:integer
mkdir(s)	สร้างไดเร็กทอรีย่อยชื่อ s	s:string

move(source,dest ,count)	คัดลอกข้อมูลจาก source ไปยัง dest จำนวน count ไบท์	source:anytype dest:anytype count:integer
rmdir(s)	ลบไดเร็กทอรี s ย่อยที่ว่างออกจากแผ่นดิสก์	s:string
runerror([errorcode])	หยุดการดำเนินการโปรแกรมและแสดงหมายเลขและรายละเอียด	errocode:integer
settextbuf(f,buf [,size])	กำหนดขนาดบัฟเฟอร์ของ f ให้เป็นแบบ buf หรืออาจกำหนดขนาดโดยตัวแปร size ก็ได้	f:text buf:array of char size:word
truncate(f)	ลบไฟล์หลังจากที่ไฟล์พอยนเตอร์ f ชี้ไฟล์อยู่	f:filepointer

ฟังก์ชันและฟังก์ชันต่อไปนี้อยู่ในยูนิตเฉพาะ ซึ่งได้แก่ graph dos crt overlay และ printer เมื่อต้องการใช้ฟังก์ชันหรือโพรซิเจอร์ใด ต้องใช้คำสั่ง uses เรียกยูนิตก่อน

ฟังก์ชันและโพรซิเจอร์ในยูนิต graph

ฟังก์ชัน	แบบ	การดำเนินการ	พารามิเตอร์
getbkcolor	integer	ส่งค่าสีพื้นปัจจุบัน (0 ถึง 15)	none
getcolor	integer	ส่งค่าสีที่ใช้ในการเขียนกราฟฟิก	none
getdefaultpallete (var pal) ลำดับสีทั้ง 16 ได้แก่ black(=0),blue green,..... .... white	palette type	ส่งแบบการแสดงสีเมื่อเข้าการกราฟฟิก palettetype= record size:byte color:array[0..maxcolor] of integer; end;	palletetype

getdrivername	string ส่งชื่อแบบของจอกราฟฟิก	none
getgraphmode	integer ส่งแบบการแสดงกราฟฟิก (mode)	none

แบบของจอ (driver) และการแสดง (mode) มีดังนี้ คือ

graphdriver	graphmode	ความละเอียด	สี	จำนวนหน้า
CGA	CGAC0(0) ถึง CGAC3(3)	320*200	4	1
	CGAHi (4)	640*200	2	1
MCGA	MCGAC0(0) ถึง MCGAC3(3)	320*200	4	1
	MCGAMed (5)	640*200	2	1
	MCGAHi (6)	640*480	2	1
EGA	EGALo (0)	640*200	16	4
	EGAHi (1)	640*350	16	2
EGA64	EGA64Lo (0)	640*200	16	1
	EGA64Hi (1)	640*350	4	1
EGAMONO	EGAMONOHi (3)	640*350	2	1
	EGAMONOHi (3)	640*350	2	2
HERC	HercMonoHi(0)	720*348	2	2
ATT400	ATT400C0(0)ถึงATT400C3(3)	320*200	2	1
	ATT400Med (4)	640*200	2	1
	ATT400Hi (5)	640*400	2	1
VGA	VGALo (0)	640*200	16	2
	VGAMed (1)	640*350	16	2
	VGAHi (2)	640*480	16	1
PC3270	PC3270Hi (0)	720*350	2	1
8514	IBM8514Lo (0)	640*480	256	1
	IBM8514Hi (0)	1024*768	256	1

getmaxcolor	word	ส่งจำนวนสีที่สามารถแสดงได้	none
getmaxmode	word	ส่งจำนวนแบบการแสดงกราฟฟิกทั้งหมด	none
getmodename	string	ส่งแบบการแสดงกราฟฟิกปัจจุบัน	none
getpaletesize	word	ส่งจำนวนสีในการแสดงกราฟฟิก	none
grapherrormsg(x)	integer	ส่ง error message ตามค่าที่ได้ให้	integer
graphresult	integer	ส่งรหัสของ error ที่เกิดขึ้น	none
installuserdriver (name,p)	word	ส่งค่าแบบการแสดงกราฟฟิกของจอที่ไม่มีการเตรียมไว้ใน turbo pascal	name:string p:pointer
installuserfont (font)	integer	ส่งค่าแบบตัวอักษรชนิดใหม่เพื่อใช้ในการแสดงตัวอักษรในกราฟฟิก	font:string
registerbgidriver (d)	integer	เรียกไฟล์ .bgi เข้าในหน่วยความจำเพื่อความสะดวกในการแสดงกราฟฟิก	d:pointer
registerbgifont(f)	integer	เรียกไฟล์ .chr เข้าในหน่วยความจำ	f:pointer

โพรซิเจอร์	การดำเนินการ	พารามิเตอร์
getmoderange (graph,low,high)	ให้ค่าการแสดงกราฟฟิกแบบละเอียดสูงสุดและต่ำสุดแก่ตัวแปร low และ high	graph:integer low,high:integer
getpalette(p)	ส่งขนาดและค่าแถบสีให้แก่ palette	p:palettetype
graphdefaults	เริ่มต้นให้ค่าการแสดงกราฟฟิกใหม่เช่น แบบเส้น ขนาดตัวอักษร การระบาย	none
setallpalette (palette)	กำหนดจำนวนและแถบสีใหม่แก่ตัวแปร palette	palette:palettetype
setbkcolor(c)	กำหนดสีพื้น	c:word
setcolor(c)	กำหนดสีที่ใช้ในการเขียนกราฟฟิก	c:word
setgraphbufsize (bufsize)	เปลี่ยนขนาดของบัฟเฟอร์ที่ใช้ระบาย	bufsize:word

setgraphmode(m)	เปลี่ยนการแสดงออกจอแบบกราฟฟิกตามที่ต้องการ ควรใช้คู่กับ getgraphmode	m:integer
setrgbpalette (colornum, red,green,blue)	เปลี่ยนแปลงการแสดงสีเฉพาะจอ คำสั่งนี้ใช้ได้เฉพาะจอ ibm-8514 หรือ vga	colornum,red green,blue:integer
setpalette (colornum,color)	กำหนดสีลำดับที่ colornum ให้เป็น color	colornum:word; color:shortint;
setwritemode (writemode)	กำหนดแบบการลากเส้น ถ้า writemode เป็น 0 ให้ลากเส้นปรกติ และลากเส้นตัดกับสีพื้นเมื่อมีการระบายที่พื้นเมื่อเป็น 1	writemode:integer

ฟังก์ชันและโพรซิเจอร์ที่อยู่ในยูนิต dos

ฟังก์ชัน	แบบฟังก์ชัน	การดำเนินการ	พารามิเตอร์
diskfree (drive)	longint	ส่งจำนวนเนื้อที่ ที่ว่างบนดิสก์ตามค่า drive (0:ไดร์ฟปัจจุบัน 1:ไดร์ฟ a)	drive:word
disksize(drive)	longint	ส่งจำนวนเนื้อที่ทั้งหมดของไดร์ฟ drive	drive:word
dosexitcode	word	ส่งค่ารหัสที่ออกจากดำเนินการในบล็อก	none
dosversion	word	ส่งค่าตัวเลขเวอร์ชัน dos	none
envcount	integer	ส่งจำนวนสตริงที่อยู่ใน environment	none
envstr(index)	string	ส่งค่าสตริงใน environment	index:integer
fexpand(path)	pathstr	ส่งชื่อไฟล์ path แบบเต็มกลับ (ไดร์ฟ ไดเร็กทอรีและนามสกุล)	path:pathstr =string[79]
fsearch(path, list)	pathstr	ส่งชื่อไฟล์ใน path จากไดเร็กทอรี list	path:pathstr list:string

getenv(envvar)	string	ส่งค่า environment อื่นๆเช่น files	envvar:string

โพรซิเจอร์	การดำเนินการ	พารามิเตอร์
exec(path,cmline)	ดำเนินการคำสั่ง cmline ใน path เช่น exec('\command.com','/c dir')	path:string cmline:string
findfirst(path,att,s) แบบของ attr มีดังนี้ readonly =$01 hidden =$02 sysfile =$04 volume =$08 directory=$10 archive =$20 anyfile =$3f	ค้นหาไฟล์ใน path ที่มีค่าแอตทริบิวต์ attr และให้ค่าไฟล์นั้นแก่ s searchrec =record fill:array[1..21] of byte; attr:byte; time,size:longint; name:string[12] end;	path:string attr:byte s:searchrec
findnext(s)	ค้นหาไฟล์ลำดับที่ถัดจากไฟล์ที่ได้จากคำสั่ง findnext ที่แล้ว หรือ findfirst	s:searchrec
fsplit(path,dir name,ext);	แยกชื่อไฟล์ของ path ออกเป็น 3 ส่วน ได้แก่ ไดเร็กทอรี ชื่อไฟล์ นามสกุล pathstr=string[79] dirstr=string[67] namestr=string[8] extstr=string[4])	path:pathstr dir:dirstr name:namestr ext:extstr

getcbreak	ตรวจสอบภาวะการกด ctrl-break	break:boolean
getdate(year,month,day dayofweek)	ส่งค่าวันที่ปัจจุบันให้พารามิเตอร์ทั้ง 4	all are word
getfattr(f,attr)	ส่งค่าแอตทริบิวต์ของไฟล์ f ให้แก่ attr	f:ไฟล์พอยน์เตอร์ attr:word
getftime(f,time)	ส่งค่าวันเวลาครั้งล่าสุดที่มีการเขียน ไฟล์ f ให้แก่ time	f:ไฟล์พอยน์เตอร์ time:longint
getintvec(intno,vector)	ส่งแอดเดรสของอินเทอร์รัพท์เวคเตอร์ ให้แก่ vector	intno:byte vector:pointer
gettime(hour,minute, second,second100);	ส่งค่าเวลาปัจจุบันให้ hour minute second และ second100	all are word
getverify(verify)	ส่งสถานะการบันทึกลงดิสก์ให้ verify	verify:bollean
intr(intno,var regs)	ดำเนินการอินเทอร์รัพท์เวคเตอร์ตาม เลข intno และเก็บค่ารีจิสเตอร์ที่ regs registers มีแบบเป็น =record case integer of 0:(ax,bx,cx,cx,bp,si,di, ds,es,flags:word); 1:(al,ah,bl,bh,cl,ch,dl, dh:byte); end;	intno:byte regs:registers
keep(exitcode)	ออกจากโปรแกรมที่กำลังดำเนินการอยู่	exitcode:word
msdos(regs)	ดำเนินการฟังก์ชั่น dos	regs:registers

packtime(dt,time)	เปลี่ยนวันเวลาปัจจุบันจากข้อมูล dt แบบ datetime ให้กับ time datetime =record year,month,day,hour, min,sec:word; end;	dt:datetime time:longint
setcbreak(break)	ติดตั้งภาวะการตรวจสอบ ctrl-break	break:bollean
setdate(year,month,day)	ตั้งค่าวันเดือนปีปัจจุบัน	all are word
setfattr(f,attr)	ตั้งค่าแอตทริบิวต์ attr ให้ไฟล์ f	f:filepointer attr:word
setftime(f,time)	ตั้งเวลา time ที่เขียนไฟล์ f ล่าสุด	f:filepointer time:longint
setintvec(intno,vector)	ตั้งค่าอินเตอร์รัพท์เวคเตอร์ intno ไปที่แอดเดรส vector	intno:byte vector:pointer
setverify(verify)	ติดตั้งการตรวจสอบการบันทึกลงดิสก์	verify:bollean
settime(hour,minute, second,sec100)	ติดตั้งค่าเวลาปัจจุบัน	all are word
swapvectors	เก็บและเรียกอินเทอร์รัพท์เวคเตอร์ก่อนและหลังการใช้คำสั่ง exec	
unpacktime(time,dt)	เปลี่ยนข้อมูลวันเวลาที่มีการเก็บในรูป longint ให้เป็นรูปของ dt	time:longint dt:datetime

ฟังก์ชันและโพรซิเจอร์ในยูนิต crt

ฟังก์ชัน	แบบฟังก์ชัน	การดำเนินการ	พารามิเตอร์
keypressed	boolean	ตรวจว่ามีการเคาะคีย์บอร์ดหรือไม่	none
readkey	char	อ่านตัวอักษรจากคีย์บอร์ด	none
wherex	byte	ส่งคอลัมน์ของเคอร์เซอร์ปัจจุบัน	none
wherey	byte	ส่งบรรทัดของเคอร์เซอร์ปัจจุบัน	none

โพรซิเจอร์	การดำเนินการ	พารามิเตอร์
assigncrt(f)	เชื่อมเท็กซ์ไฟล์เข้ากับจอ	f:text
clreol	ลบตัวอักษรบนจอภาพจากตำแหน่งที่เคอร์เซอร์อยู่จนสิ้นสุดบรรทัด	none
clrscr	ลบจอภาพทั้งจอภาพและย้ายเคอร์เซอร์ไปยังตำแหน่งมุมซ้ายบนสุด	none
delay(ms)	หน่วงเวลา ms milliseconds	ms:word
delline	ลบบรรทัดที่มีเคอร์เซอร์อยู่	none
gotoxy(x,y)	ย้ายเคอร์เซอร์ไปยังบรรทัดที่คอลัมน์ x บรรทัดที่ y	x:byte y:byte
highvideo	แสดงตัวอักษรบนจอแบบเข้ม	none
insline	แทรกบรรทัดว่างเปล่าที่ตำแหน่งเคอร์เซอร์	none
lowvideo	แสดงตัวอักษรบนจอแบบจาง	none
normvideo	แสดงตัวอักษรบนจอแบบธรรมดา	none
nosound	ปิดเสียงจากลำโพง	none
sound(Hz)	ส่งเสียงความถี่ Hz ออกลำโพง	Hz:word
textbackground (color)	เปลี่ยนสีพื้นบนจอ (color=0..7) color=(black...lightgray)	color:byte

textmode(mode)	เปลี่ยนการแสดงเท็กซ์ตามโหมดต่างๆ const bw40 = 0 (40*25 b/w on color adapter) bw80 = 2 (80*25 b/w on color adapter) mono = 7 (80*25 on monochrome adapter) co40 = 1 (40*5 color on color adapter) font8*8 = 256 (for ega/vga 43 and 50 line)	mode:integer
window (X_1,Y_1,X_2,Y_2)	กำหนดขอบเขตของการแสดงตัวอักษร โดยให้ (X_1,Y_1) เป็นตำแหน่งมุมซ้ายบน และให้ (X_2,Y_2) เป็นตำแหน่งมุมขวาล่าง	all are byte

โพรซิเจอร์ในยูนิต printer

lst: นำข้อความหรือตัวแปรออกเครื่องพิมพ์โดยคำสั่ง write(lst,....)

โพรซิเจอร์ในยูนิต overlay

การใช้ยูนิตเป็นการแยกโปรแกรมเป็นส่วนๆและทำการคอมไพล์เป็นภาษาเครื่องเพื่อความสะดวก ในกรณีที่มีการใช้ยูนิตมากๆ จะเกิดปัญหาเรื่องหน่วยความจำ กล่าวคือ หน่วยความจำไม่พอ วิธีการป้องกันปัญหานี้คือการ ใช้ยูนิตแบบ overlay การสร้างยูนิตชนิดนี้สามารถเขียนได้ดังนี้คือ

```
{$o+,f+}          (* ส่วนที่เพิ่มจากยูนิตธรรมดา *)
unit test;
{$f+}             (* ส่วนที่เพิ่มจากยูนิตธรรมดา *)
interface
procedure plus(i,j:integer;var k:integer);
implementation
  procedure plus;
  begin
     k:=i+j
  end;
end.
```

เมื่อสร้างยูนิต test เสร็จ การเรียกยูนิต test ใช้สามารถเขียนได้ดังนี้

```
program ok;
{Sf+}
uses overlay,test;
var v:integer;
{$o test}                (* เรียกยูนิต overlay ชื่อ test *)
begin
   ovrinit('ok.ovr');    (* ใช้ชื่อเดียวกับชื่อไฟล์ (ok.pas) *)
   plus(3232,33,v);
   writeln(v);
end.
```

เมื่อดำเนินการโปรแกรมนี้แล้ว ในไดเรกทอรีจะปรากฏไฟล์ชื่อ ok.ovr และ ok.exe ในกรณีที่มีการเรียกใช้ยูนิตอื่น เช่น ยูนิต test1 ถ้าหากว่าหน่วยความจำไม่เพียงพอ ยูนิต test จะถูกนำออกจากหน่วยความจำ หลังจากนั้นจึงเรียกยูนิต test1 เข้ามาในหน่วยความจำ

โพรซิเจอร์	การดำเนินการ	พารามิเตอร์
ovrinit(filename)	ตั้งค่าตัวจัดการโอเวอร์เลย์ และเปิดไฟล์โอเวอร์เลย์	filename:string
ovrinitems	เรียกไฟล์โอเวอร์เลย์เข้าไปใน ems	none
ovrsetbuf(size)	ตั้งขนาดบัฟเฟอร์ของโอเวอร์เลย์	size:longint
ovrclearbuf	ลบบัฟเฟอร์ของโอเวอร์เลย์ออก	none

ฟังก์ชัน	แบบฟังก์ชัน	การดำเนินการ	พารามิเตอร์
overgetbuf	longint	ส่งขนาดบัฟเฟอร์ของโอเวอร์เลย์ปัจจุบัน	none

คำสงวนเฉพาะใน turbo pascal

absolute: ใช้สำหรับประกาศตัวแปร โดยสามารถกำหนดตำแหน่งหน่วยความจำได้ เช่น

```
var crtmode:byte absolute $0040:0049;
```

external: ใช้สำหรับนิยามโพรซิเจอร์หรือฟังก์ชันที่เขียนขึ้นโดยภาษาแอสเซมบลีจากไฟล์ ดังตัวอย่าง

```
ex) {$l plus.obj}     (* เรียกไฟล์ plus.obj ซึ่งมีโพรซิเจอร์ add อยู่ *)
    procedure  add(i,j:integer;var k:integer):external;
```

inline: คำสั่งนี้ใช้คล้ายกับโพรซิเจอร์หรือฟังก์ชันที่บรรจุรหัสคำสั่ง machine code หรือภาษาเครื่องภายในวงเล็บโดยมีเครื่องหมาย '/' คั่นระหว่างไบท์

```
ex) procedure disableinterrupts: inline($fa)        (* cli *)
    procedure enableinterrupts:  inline($fb)        (* sti *)
    function longmul(x,y:integer):longint; (* ฟังก์ชันการคูณเลข 2 จำนวน *)
    inline(
       $5a/                                 pop ax:pop x
       $58/                                 pop dx;pop y
       $f7/$ea);                            imul dx;dx:ax=x*y
```

interrupt: เป็นคำสั่งสร้างอินเทอร์รัพท์เวคเตอร์ขึ้นเองโดยมีลักษณะตอนต้นดังที่เห็น

```
ex) procedure testint(flags,cs,ip,ax,bx,cx,dx,si,di,ds,es,bp:word);
        interrupt;
```

โดยเก็บโพรซิเจอร์ testint ด้วยคำสั่ง setintvec(intno,@testint)

shl: โอเปอร์เรเตอร์ สำหรับเลื่อนบิทไปทางซ้าย โดยที่ i shr j คือการเลื่อนบิทของ i ไปทางซ้าย จำนวน j บิท

shr: โอเปอร์เรเตอร์ สำหรับเลื่อนบิทไปทางขวา โดยที่ i shr j คือการเลื่อนบิทของ i ไปทางขวา จำนวน j บิท

< การสร้างฟังก์ชันใหม่สำหรับการคำนวณทางคณิตศาสตร์ >

ในภาษา Pascal ฟังก์ชันที่ใช้ในการคำนวณคณิตศาสตร์ที่ซับซ้อนสามารถสร้างจากฟังก์ชันที่มีอยู่ได้ สำหรับฟังก์ชัน sgn มีค่าเป็น 1 เมื่อ x มีค่าเป็นบวก

เป็น 0 เมื่อ x มีค่าเป็นศูนย์

และเป็น -1 เมื่อ x มีค่าเป็นลบ

ฟังก์ชัน	นิพจน์
sin^{-1}	arctan(x/sqrt(-x*x+1))
cos^{-1}	-arctan(x/sqrt(-x*x+1))+pi/2
sec^{-1}	arctan(sqrt(x*x-1)+(sgn(x)-1)*pi
cosec^{-1}	arctan(1/sqrt(x*x-1))+(sgn(x)-1)*pi
cot^{-1}	-arctan(x)+pi/2
sinh	(exp(x)-exp(-x))/2
cosh	(exp(x)+exp(-x))/2
tanh	-exp(-x)/(exp(x)+exp(-x))*2+1
sech	2/(exp(x)+exp(-x))
cosech^{-1}	2/(exp(x)-exp(-x))
coth^{-1}	exp(-x)/(exp(x)-exp(-x))*2+1
sinh^{-1}	ln(x+sqrt(x*x+1))
cosh^{-1}	ln(x+sqrt(x*x-1))
tanh^{-1}	ln((1+x)/(1-x))/2
sech^{-1}	ln((sqrt(-x*x+1)+1)/x)
cosech^{-1}	ln((sgn(x)+sqrt(x*x+1)+1)/x)
coth^{-1}	ln((x+1)/(x-1))/2
LOGa(B)	ln(B)/ln(a)
A^{b}	exp(b*ln(A))

ภาคผนวก ค

การสั่งงานคอมไพเลอร์ (Compiler Directives)

Turbo Pascal สามารถสั่งงานคอมไพเลอร์ได้หลายวิธี ดังที่เห็น

- ในโปรแกรม โดยใช้ {$directive}
- ใน integreated environment (ในเมนู Options/compiler)
- Command Line สำหรับคอมไพเลอร์ TPC โดยมีแบบฟอร์มดังนี้

  .>tpc program /$<directive>
- ในไฟล์ tpc.cfg หรือ turbo.tp

Compiler Directive อยู่ในวงเล็บปีกกาโดยจะเริ่มต้นด้วยตัวอักษร $ และตามด้วยชื่อหรือสัญญลักษณ์ซึ่งอาจเป็นอักษรเดียวหรือหลายตัวอักษร เพื่อสั่งงานคอมไพเลอร์

Compiler Directive มีอยู่ 3 ชนิดด้วยกัน

1. แบบสวิทช์ มีลักษณะคล้ายสวิทช์เปิดปิดโดยใช้เครื่องหมาย +,- หลัง directive เช่น {$B+} {$R-}

2. แบบมีพารามิเตอร์โดยมีพารามิเตอร์ตามหลังชื่อ directive ซึ่งอาจเป็นชื่อไฟล์หรือขนาดของหน่วยความจำ เช่น {$I TYPE.INC} {$M 65520, 8192, 655360}

3. แบบมีเงื่อนไข ใช้ควบคุมการคอมไพล์โปรแกรม โดยสัญญลักษณ์เงื่อนไขที่ได้วางไว้ในลักษณะของ if และ else และเงื่อนไขก็คือการนิยามสัญญลักษณ์เงื่อนไขไว้หรือไม่

ตัวอย่างคือ

```
{$DEFINE Debug}
{$IFDEF Debug}
      ... (ข้อความที่ต้องการให้คอมไพล์)
{$ENDIF}
```

หากสัญญลักษณ์เงื่อนไข ไม่เป็นจริงจะไม่คอมไพล์ข้อความดังกล่าว

และในการเขียน Compiler Directives แบบมีตัวแปรต้องมีช่องว่างหรือเว้นวรรคระหว่างชื่อ Directive กับตัวแปรต่างๆ ส่วนแบบที่เหลือห้ามมีเว้นวรรค

Compiler Directive แบบสวิทช์นี้มี 2 ชนิดด้วยกันนั่นคือ

1. ชนิด global จะต้องอยู่หลังโปรแกรม หรือ ก่อนยูนิต

2. ชนิด local มีลักษณะในทางตรงกันข้ามกับชนิด global นั่นคือ อยู่ที่ใดก็ได้ซึ่งจะมีผลเฉพาะส่วนที่อยู่หลัง Compiler Directive จนกว่าจะมีการเปลี่ยนแปลง

Compiler Directive แบบสวิทช์นี้ จะสามารถรวมอยู่ใน Comment (วงเล็บปีกกา) เดียวกันได้โดยคั่นด้วย จุลภาค ',' ดังเช่น {$B+,R+,$+} โดยใช้คอมมาคั่น

< การสั่งงานคอมไพเลอร์แบบสวิทช์ >

Align Data

แบบ {$A+} หรือ {$A-} ค่าปรกติ {$A+} หรือ on ใน pull Down Menu

ชนิด global

การติดตั้ง directive ในเมนู Options/Compiler/Align Data

คำสั่งนี้จะทำให้การบันทึกข้อมูลที่ตำแหน่งหน่วยความจำเลขคู่ ซึ่งจะทำให้อ่านข้อมูลได้เร็วขึ้น (คำสั่งนี้มีผลเฉพาะ cpu เบอร์ 8086 และไม่มีผลต่อ cpu เบอร์ 8088)

Boolean Evaluation

แบบ {$B+} หรือ {$B-} ค่าปรกติ {$B-} ชนิด local

Compiler Directive นี้จะทำหน้าที่เป็นเหมือนสวิทช์ของการทำงาน 2 ลักษณะในทางคำนวณหรือหาค่าทางตรรกะหรือ Boolean โดยใช้ and และ or

ในสภาวะ {$B+} คอมไพเลอร์จะกระทำการตรวจสอบนิพจน์ทาง Boolean ทุกๆนิพจน์ แต่ในสภาวะ {$B-} จะตรวจสอบนิพจน์ทางบูลีนอย่างสั้น เมื่อทราบคำตอบแล้วก็จะหยุดทำนิพจน์อื่นที่เกี่ยวข้องในข้อความที่ต้องการหาค่าทางตรรกะนั้น โดยอาศัยความจริงที่ว่า

A and B หากพบว่า A เป็นเท็จ ก็จะสรุปได้ว่า A and B เป็นเท็จเช่นกัน

Debug Information

แบบ {$D+} หรือ {$D-} ค่าปรกติ {$D+} ชนิด global

โดยปรกติ ข้อมูลสำหรับการแก้ไขข้อผิดพลาดนี้จะเก็บตารางแต่ละบรรทัดของตัวโปรแกรมเมื่อเกิด runtime error ในโปรแกรมหรือยูนิต ถ้าหากตั้ง {$D+} ไว้ จะมีการระบุตำแหน่งบรรทัดของข้อความในโปรแกรมที่ทำให้เกิดข้อผิดพลาด ในกรณีที่มีสภาวะ {$D+} เมื่อคอมไพล์เป็นยูนิตจะทำให้มีเนื้อที่มากกว่าคอมไพล์ในสภาวะ {$D-}

Emulation

แบบ {$E+} หรือ {$E-} ค่าปรกติ {$E+} ชนิด global

ใน Turbo Pascal มีโปรแกรมจำลองตัวประมวลผลทางคณิตศาสตร์ 8087 ถ้าหากอยู่ในสภาวะ {$N+,E+} และไม่มี 8087 ก็จะนำโปรแกรมจำลองมาช่วยคำนวณ แต่ถ้าอยู่ในสภาวะ {$N+,E-} จะต้องมี 8087 ด้วย

Force Far Call

แบบ {$F+} หรือ {$F-} ค่าปรกติ {$F-} ชนิด local

คำสั่ง {$F+} โดยปรกติจะใช้สำหรับเรียกยูนิตประเภทโอเวอร์เลย์ ซึ่งต้องเขียน {$F+} ที่ต้นโปรแกรมหลักและที่ต้นยูนิต ตัวอย่างการใช้งานโอเวอร์เลย์ยูนิตที่ภาคผนวก ข

Input/Output Checking

แบบ {$I+} หรือ {$I-} ค่าปรกติ {SI+} ชนิด local

การตั้งค่าในเมนู Options/compiler/I/O checking

ดูท้ายบทที่ 8 คำสั่งนี้ใช้สำหรับตรวจการอ่านและบันทึกข้อมูล เช่น read reset เป็นต้น

Link Buffer

แบบ {$L+} หรือ {$L-} ค่าปรกติ {$L+} ชนิด global

การตั้งค่าในเมนู Options/Compiler/Line buffer

คำสั่งนี้ต้องใช้ในขณะที่อยู่ในสภาวะ {$D+} เท่านั้น การดำเนินการคำสั่งนี้คือ สามารถตรวจสอบค่าตัวแปรในยูนิต

Numeric Processing

แบบ {$N+} หรือ {$N-} ค่าปรกติ {$N-} ชนิด global

การตั้งค่าในเมนู Options/Compiler/Numeric Processing

เป็นคำสั่งคำนวณและแสดงเลขจำนวนจริง ในกรณีที่เป็น {$N-} จะคำนวณได้ตัวเลขได้เฉพาะแบบ real ถ้าเป็น {$N+} จะเป็นการคำนวณใช้ตัวประมวลผลรวมทางคณิตศาสตร์ 8087

ex) {$N+} begin writeln(Pi); end {$N-} begin writeln(pi); end

ผลออกจอภาพ 3.14159265358979E+0000 ผลออกจอภาพ 3.1415926536E+00

แบบข้อมูลที่สามารถใช้ในสภาวะ {$N+} มีดังนี้คือ

แบบ	ขอบเขต	จำนวนหลักที่แสดง
single	$-1.5*10^{-45}$ .. $3.4*10^{38}$	7-8
double	$-5.0*10^{-324}$ .. $1.7*10^{308}$	15-16
extended	$-3.4*10^{-4932}$ .. $1.1*10^{4932}$	19-20
comp	$-2^{64}+1$ .. $2^{64}-1$	19-20

Overlay Code Generation

แบบ {$O+} หรือ {$O-} ค่าปรกติ {$O-} ชนิด global

การตั้งค่าในเมนู Options/Compiler/Overlays Allowed

ยูนิตที่กำหนด {$O+} ไว้ที่ต้นโปรแกรมคือ ยูนิตที่สามารถเรียกมาใช้งานแบบโอเวอร์เลย์ได้

Range-Checking

แบบ {$R+} หรือ {$R-} ค่าปรกติ {$R-} ชนิด local

การตั้งค่าในเมนู Options/Compiler/Range checking

ในภาวะ {$R+} ทุกนิพจน์ที่เกี่ยวกับ array และ index ของ string จะมีการตรวจสอบว่าอินเด็กซ์อยู่ในขอบเขตที่กำหนดไว้หรือไม่ ถ้าพบว่าไม่อยู่ในช่วงที่ได้กำหนดไว้ตอนต้นจะเกิด error

Stack Overflow Checking

แบบ {$S+} หรือ {$S-} ค่าปรกติ {$S+} ชนิด local

การตั้งค่าในเมนู Options/Compiler/Stack checking

เมื่อคอมไพล์ด้วยภาวะ {$S+} คอมไพเลอร์จะตรวจสอบว่า สแต็กที่ใช้เก็บค่าตัวแปรเฉพาะที่ใช้ในโพรซีเจอร์ หรือฟังก์ชันนั้นว่าเพียงพอหรือไม่ ถ้าหากไม่เพียงพอก็จะออกจากโปรแกรมและแสดงข้อความ runtime error

Var-String Checking

แบบ {$V+} หรือ {$V-} ค่าปรกติ {$V+} ชนิด local

การตั้งค่าในเมนู Options/Compiler/Var-String Checking

directive นี้ใช้ตรวจสอบชนิดของตัวแปรสตริงโดยที่ ในภาวะ {$V+} เมื่อมีการเรียกโพรซีเจอร์โดยผ่านพารามิเตอร์ซึ่งเป็นแบบสตริง พารามิเตอร์ทางฝ่ายส่งและรับต้องเป็นสตริงที่มีความยาวเท่ากัน

< แบบมีพารามิเตอร์ >

Include File

แบบ {$Iชื่อไฟล์} ชนิด local

การตั้งค่าในเมนู Options/Directories/Include directories

ไดเร็กทีฟนี้จะทำให้คอมไพเลอร์ไปเปิดและอ่านไฟล์ที่ตามหลัง directive เข้ามาทำในรวมกับโปรแกรมหลักหากไม่ระบุนามสกุลของไฟล์นั้น คอมไพเลอร์จะถือว่าเป็นไฟล์ .PAS และถ้าไม่เจาะจงไดเร็กทอรี่ คอมไพเลอร์จะค้นหาไฟล์ตามชื่อดังกล่าวในไดเร็กทอรี่ปัจจุบัน และไดเร็กทอรี่ที่ตั้งใน Options/Directories/inculde directories หรือไดเร็กทอรี่ที่ตั้งใน option/I สำหรับ TPC command line

Link Object File

แบบ {$L ชื่อไฟล์ .OBJ}

ชนิด local

การตั้งค่าในเมนู Option/Directories/Object Directories

มีลักษณะคล้ายคลึงกับ $I แต่เป็นการสั่งงานให้เชื่อมไฟล์ .obj เข้ากับโปรแกรมหรือยูนิตที่จะคอมไพล์ โดยฟังก์ชันหรือโพรซีเจอร์ของโปรแกรมย่อยนี้จะประกาศเป็น external ในยูนิตหรือโปรแกรมที่เรียกใช้ หากไฟล์นั้นไม่ได้ระบุไดเร็กทอรี่ Turbo Pascal จะค้นหาจากไดเร็กทอรีปัจจุบัน และตามด้วยไดเร็กทอรีที่ตั้งไว้ในเมนู Options/Directories/Object directories หรือค้นหาใน Option/o บน TPC command line

ตัวอย่างการเชื่อมโยงกับภาษา Assembly

```
program addt;
var d:integer;
  procedure plus(a,b : integer; var c : integer); external;
  {$L test}
begin
  plus(100,200,d);
  writeln('a+b = ',d);
end.
```

สำหรับโพรซิเจอร์ในภาษาแอสเซมบลีสามารถเขียนได้ดังนี้

```
code  segment byte public
      assume cs:code
      public plus
;procedure plus(a,b:integer; var c : integer); external;
plus  proc    near
      push    bp
      mov     bp,sp
      mov     ax,[bp+10]              ; A:integer
      add     ax,[bp+8]               ; A:=A+B
      les     di,dword ptr [bp+4]     ; es = segment of variable c
                                      ; di = offset of variable c
      stosw                           ; store ax to address pointed by
                                      ; es:[di]
      pop     bp
      ret     4
plus  endp
code  ends
      end
```

เมื่อคอมไพล์ภาษาแอสเซมบลีจะได้ไฟล์ชื่อ test.obj

การใช้ภาษาแอสเซมบลีเชื่อมกับภาษา Pascal มีหลักดังนี้

- ส่วนประกาศโพรซีเจอร์หรือฟังก์ชันมีคำ extermal ต่อท้าย

- ใช้ directive $L ตามด้วยชื่อไฟล์โปรแกรมย่อย (test.obj)

- ใช้ชื่อเซ็กเมนต์ของคำสั่งว่า code

- ชื่อโพรซีเจอร์หรือฟังก์ชันในภาษา Pascal ต้องอยู่ในคำสั่ง Public และตรงกับชื่อ Proc ในภาษา Assembly

- พิจารณาลักษณะการเรียกโพรซีเจอร์หรือฟังก์ชันว่าเป็น near หรือ far และโปรแกรมย่อย Assembly กับ Pascal อยู่ในเซ็กเมนต์เดียวกันหรือไม่

- พิจารณาพารามิเอตร์ที่ส่งมาจาก Pascal และที่จะส่งกลับ

Memory Allocation Sizes

แบบ {$M stacksize,heapmin,heapmax} ค่าปรกติ {$M 16384,0,655360}

ชนิด global

การตั้งค่าในเมนู Obtions/Compiler/memory size

directive นี้ใช้เนื้อที่ในหน่วยความจำของตัวแปรในโปรแกรม ขนาดของสแต็กจะต้องเป็นตัวเลขจำนวนเต็มที่อยู่ในช่วง 1024 (1k) จนถึง 65520 สำหรับค่าเริ่มต้นของ heap อยู่ในช่วง 0-655360 ค่าสิ้นสุดของ heap อยู่ในช่วงค่าเริ่มต้นของ heap ถึง 655360 Compiler Directive นี้จะไม่มีผลเมื่อใช้ในยูนิต

Overlay Unit Name

แบบ {$O unitname} ชนิด local

คำสั่งนี้จะดำเนินการยูนิต (unitname) ให้เป็นแบบโอเวอร์เลย์ ตำแหน่งของคำสั่งนี้ต้องวางหลังคำสั่ง uses ชื่อยูนิตดังตัวอย่าง

```
program test;
{$F+}
uses test1;
{$O test1}
```

เมื่อดำเนินการคอมไพล์โปรแกรมนี้ จะได้ไฟล์ test1.ovr และ test1.exe

< แบบมีเงื่อนไข >

วัตถุประสงค์ของ Compiler Derective แบบนี้คือ คอมไพล์ส่วนที่ต้องการคอมไพล์

ลักษณะทั่วไปของการเขียน Compiler Derective คือ

```
{$IFDEF condition}
   บล็อก S1
{$ELSE}
   บล็อก S2
{$ENEIF}
```

จากตัวอย่างข้างบน ถ้ามีการกำหนดสัญญลักษณ์เงื่อนไข condition ไว้ก่อนหน้านี้ ก็จะคอมไพล์บล็อก S1 ถ้าไม่มีการกำหนด จะคอมไพล์บล็อก S2 สัญญลักษณ์เงื่อนไขไม่ใช่ตัวแปรแบบบูลีน

วิธีการสร้างสัญญลักษณ์เงื่อนไข สามารถสร้างโดยคำสั่ง $DEFINE condition ดังตัวอย่าง

```
{$DEFINE condition}
{$IFDEF condition}
    write('This Condition is');
    writeln('OK!');
{$ELSE}
    writeln('not OK');
{$ENDIF}
```

คำสั่ง Compiler Directive อื่นได้แก่

{$UNDEF} ชื่อสัญญลักษณ์เงื่อนไข มีผลตรงข้ามกับ {$DEFINE}

{$IFNDEF} ชื่อสัญญลักษณ์เงื่อนไข มีผลตรงข้ามกับ {$IFDEF}

{$IFOPT} Compiler Derective แบบสวิทช์ ตรวจดูสภาวะปัจจุบันเช่น {$IFOPT N+}

สำหรับสัญญลักษณ์เงื่อนไขที่สามารถนำมาใช้ได้โดยไม่ต้องกำหนดมีดังนี้คือ

VER_{50} เป็นสัญญลักษณ์เงื่อนไขจริง ถ้าใช้ Turbo $VER_{5.0}$

MSDOS เป็นสัญญลักษณ์เงื่อนไขจริง ถ้าดำเนินการบน MSDOS

CPU_{86} เป็นสัญญลักษณ์เงื่อนไขจริง ถ้าใช้ CPU ตระกูล 8086

CPU87 เป็นสัญญลักษณ์เงื่อนไขจริงถ้าใช้ตัวประมวลผลร่วมทางคณิตศาสตร์เบอร์ 8087

ภาคผนวก ง

โปรแกรมช่วยงานอื่นๆ (utility)

TURBO.TPL ยูนิตมาตรฐาน ประกอบด้วยยูนิต system crt dos printer และ overlay

TURBO.HLP โปรแกรมแสดงรายละเอียดวิธีการใช้งาน Turbo Pascal ต้องเก็บไว้ที่ไดรฟ์เดียวกับ TURBO.EXE

GRAPH.TPU ยูนิตเก็บคำสั่งการเขียนกราฟฟิก

GRAPH3.TPU ยูนิตเก็บคำสั่งกราฟฟิกของ Version 3.0

TURBO3.TPU ยูนิตเก็บคำสั่งที่ใช้ใน Version 3.0 ซึ่งมีคำสั่ง read(kbd,..) longfilesize longfileseek และ longseek

TPUMOVER.EXE โปรแกรมสำหรับเคลื่อนย้ายไฟล์ tpu เข้าออกจากไฟล์.tpl

TINST.EXE โปรแกรมติดตั้งค่าเริ่มต้นต่างๆให้ Turbo Pascal ซึ่งมีลักษณะการใช้งานเช่นเดียวกับการใช้เมนู

UPGRADE.EXE โปรแกรมแก้ไขโปรแกรม Version 3.0 ให้ใช้กับ Version ปัจจุบันได้

MAKE.EXE โปรแกรมจัดการแก้ไขโปรแกรมบน DOS

GREP.COM โปรแกรมค้นหาคำในไฟล์ต่างๆ

TOUCH.COM โปรแกรมเปลี่ยนวันที่ของไฟล์เป็นวันที่ปัจจุบัน ใช้ในกรณีที่ต้องการคอมไพล์ไฟล์ โดยที่ไม่ได้แก้ไขไฟล์นั้น

TPCONFIG.EXE โปรแกรมส่งแบบการคอมไพล์ต่างๆระหว่างไฟล์ TURBO.TP กับไฟล์ TPC.CFG

BINOBJ.EXE โปรแกรมเปลี่ยนไฟล์ .com ของ Version 3.0 เป็นไฟล์ .OBJ

TPC.EXE เป็นโปรแกรมดำเนินการโดยใช้งานบน DOS

README ไฟล์เก็บข้อมูลการใช้งานของต่างๆ

README.COM โปรแกรมแสดงไฟล์ README (ไฟล์แสดงรายละเอียด Turbo Pascal)

*.BGI ไฟล์เก็บแบบการแสดงกราฟฟิกของจอแต่ละชนิดเช่น CGA MCGA เป็นต้น

*.CHR ไฟล์เก็บแบบของตัวอักษรแต่ละชนิดเช่น TRIP.CHR

การสร้างยูนิตและโปรแกรมช่วยงานต่างๆ

< การสร้างและใช้งานยูนิต >

ก่อนอื่นควรเข้าใจถึงการสร้างและวิธีการใช้ยูนิตเสียก่อน ยูนิตคือ โพรซิเจอร์ที่คอมไพล์เป็นภาษาเครื่อง การสร้างยูนิตสามารถสร้างได้ดังที่เห็นข้างล่าง (ต้องตั้งชื่อไฟล์เป็นชื่อเดียวกับยูนิต)

```
unit test;
inteface
(* ใส่แบบข้อมูลหรือตัวแปรแบบ global ที่นี่ ถ้าจำเป็น *)
procedure plus(i,j:integer;var k:integer);
(* ใส่โพรซิเจอร์หรือฟังก์ชันอื่นๆ ในกรณีที่มีมากกว่าหนึ่งโพรซิเจอร์ *)
implementation
procedure plus;
(* ประกาศตัวแปรแบบ local ที่นี่ ถ้าจำเป็น *)
  begin
     k:=i+j
  end;
(* สร้างโพรซิเจอร์หรือฟังก์ชันอื่นที่นี่ ในกรณีที่มีมากกว่า 1 โพรซิเจอร์หรือฟังก์ชัน *)
end.
```

เมื่อคอมไพล์ลงดิสก์ จะปรากฏยูนิตชื่อ test.tpu หลังจากนั้น ก็สามารถใช้โพรซิเจอร์ plus ได้ดังที่เห็นข้างล่าง

```
ex) program testyou;
    uses test;     (* คำสั่งเรียกยูนิต test *)
    var L:integer;
    begin
      plus(10,20,L);
      writeln(L)
    end.
```

< TPUMOVER >

เมื่อสร้างยูนิตแล้ว อาจเก็บไว้เป็นไฟล์.TPU ธรรมดา หรือเก็บใส่ไฟล์ TURBO.TPL ก็ได้ วิธีการเก็บใส่ไฟล์ TURBO.TPL สามารถเก็บโดยใช้คำสั่ง

>TPUMOVER TURBO TEST

บนจอจะปรากฏเป็น

```
                    Turbo Pascal Unit Librarian
                           Version 5.0

 A:\TURBO.TPL                              A:\TEST.TPU
 Unit      Code  Data  Syms Uses           Unit      Code  Data  Syms Uses

 SYSTEM   21332   664  4036                TEST       138 50000   568
 OVERLAY   1586    10   696
 CRT       1555    20  1651
 DOS       1589     6  4143
 PRINTER     37   256   276

 File size:     42 K                       File size:      1 K
 Drive A:      100 K free                  Drive A:      100 K free
F1-Help F2-Save F3-New F4-Info F6-Switch +-Mark INS-Copy Del-Delete Esc-Quit
```

ขั้นตอนการเก็บยูนิต (TEST) เข้าใน TURBO.TPL สามารถเขียนได้ดังนี้

1 กด F6 เพื่อย้ายเคอร์เซอร์ไปที่หน้าต่างขวา

2 กด + เพื่อเลือกไฟล์ TEST

3 กด INS เพื่อย้ายยูนิต

4 กด F6 เพื่อย้ายเคอร์เซอร์กลับมาที่หน้าต่างเดิม

5 กด F_2 เพื่อบันทึก TURBO.TPL

6 กด Esc เพื่อกลับสู่ DOS

หมายเหตุ

ในกรณีที่ต้องการย้ายยูนิตออก ให้เปลี่ยนจาก Ins เป็น Del

< MAKE >

การใช้คำสั่ง MAKE เริ่มจากการเขียนไฟล์เพื่อใช้งานโดยมีหลักการทั่วไปในการเขียนไฟล์ MAKE ดังนี้

```
TARGET [TARGET..] : SOURCE [SOURCE]
/ เขียนคอมเมนต์ที่นี่โดยอยู่ภายในเครื่องหมาย '/' /
 [COMMAND] (* ต้องย่อหน้าอย่างน้อย 1 ช่องไฟ *)
 [COMMAND] (* คำสั่งใน COMMAND จะใช้คำสั่งใดๆก็ได้เช่น cd path ฯลฯ *)
 ...
```

เมื่อสร้างไฟล์เรียบรอย ปรกติจะเก็บไฟล์ในชื่อของ MAKEFILE และใช้คำสั่ง

MAKE ได้ทันที

หรือจะใช้ไฟล์ที่ตั้งชื่อเอง เช่น testmake.mak และใช้คำสั่ง

MAKE -ftestmake.mak ก็ได้

การเขียนไฟล์ MAKE สามารถเขียนได้หลายวิธีดังต่อไปนี้

1 การเขียนแบบเจาะจง

```
ex)  test1.obj: test1.asm
      tasm test1.asm test1.obj

     mymake.tpu wymake.pas
      tpc mytest /tc:\tp5\bin
```

จากตัวอย่างข้างบน เป็นการสร้างไฟล์ test1.obj จาก test1.asm โดยคำสั่ง tasm (Turbo Assembly) และสร้างไฟล์ mytest.tpu และเก็บไฟล์ดังกล่าวไว้ที่ไดร์ฟ c ในซับไดเร็กทอรี tp5\bin ตามลำดับ

2) การเขียนแบบทั่วไป

```
ex)     .asm.obj:
         tasm $*.asm,$*.obj;
```

จากตัวอย่างนี้ จะเป็นการสร้างไฟล์.obj ขึ้นจากไฟล์.asm ที่มีอยู่ทุกไฟล์

ที่ส่วนต้นของ COMMAND ยังสามารถกำหนดแบบของการใช้คำสั่ง MAKE ได้ดังต่อไปนี้

@ เป็นคำสั่งให้ดำเนินการคำสั่ง MAKE โดยไม่แสดงรายละเอียดออกจอ

-num เป็นการกำหนดตัวเลข exit code ในกรณี exit code เกินตัวเลขที่กำหนดจะเลิกดำเนินการคำสั่ง MAKE

\- ไม่มีการตรวจ exit code

3) การเขียนโดยใช้ Macros

ในกรณีที่มีการใช้คำสั่ง หรือ path บ่อยๆ ผู้ใช้สามารถเขียนเป็นเสมือนกับตัวแปรดังตัวอย่าง

```
ex)  TURBO = c:\TP5\bin
     test1.exe:test2.pas test3.tpu test4.tpu
       tpc test1 / t$(TURBO)
```

ความหมายก็คือให้สร้างไฟล์ test1.exe และเก็บไว้ที่ c:\tp5\bin

หรือจะกำหนดที่คำสั่ง MAKE ก็ได้ดังตัวอย่าง

```
     MAKE -DTURBO=C:\TP5\bin
```

สัญญลักษณ์ต่างๆที่ใช้ใน Macros

1) $d สัญญลักษณ์ตรวจว่ามีการกำหนด Macros แล้วหรือไม่

```
ex)  !if !$d(TURBO)            (* ถ้ายังไม่มีการกำหนด TURBO *)
     TURBO=c:\tp5\bin          (*กำหนดให้ TURBO=c:\tp5\bin *)
     !endif
```

2) $* สัญญลักษณ์แทนไฟล์ TARGET โดยไม่มีนามสกุลดังตัวอย่าง

```
ex)  test1.exe:test1.pas test2.tpu
       tpc $*
```

ความหมายก็คือ คอมไพล์ไฟล์ test1.pas เป็นไฟล์ test1.exe โดยใช้ test2.tpu

3) $< สัญญลักษณ์แทนไฟล์ TARGET โดยใช้ชื่อเต็ม

4) $: สัญญลักษณ์แทนไฟล์ PATH ของไฟล์ TARGET

5) $. สัญญลักษณ์แทนไฟล์ TARGET โดยไม่มี PATH

6) $& สัญญลักษณ์แทนไฟล์ TARGET โดยไม่มี PATHและนามสกุล

การสั่งงานคำสั่ง MAKE (Directive)

การสั่งงานคำสั่ง MAKE มีดังนี้คือ

1) !include เป็นคำสั่งอ่านไฟล์ ซึ่งมีรูปแบบดังนี้

!include "ชื่อไฟล์"

หรือ !include < ชื่อไฟล์ >

2) !if คำสั่งตรวจ นิพจน์ที่ตามมาเป็นจริงหรือเท็จ

3) !else คำสั่งเช่นเดียวกับ else

4) !elif คำสั่งเช่นเดียวกับ else if

5) !endif คำสั่งสิ้นสุดการใช้เงื่อนไข !if

6) !error คำสั่งพิมพ์ข้อความหลัง !error ในกรณีที่คำสั่ง MAKE ดำเนินการไม่ได้

7) !undef คำสั่งกำหนด Macros โดยสามารถเขียนเป็น !undef macro_name

เครื่องหมายต่างๆที่สามารถใช้ในไฟล์ MAKE มีดังนี้คือ

เครื่องหมายสำหรับนิพจน์เดียว

– ลบ

~ สลับค่าในแต่ละบิตของข้อมูล

! ผลเป็นตรงข้าม (แบบบูลีน)

เครื่องหมายสำหรับนิพจน์ 2 นิพจน์

+ บวก	^ คำสั่ง xor แบบเปรียบเทียบทีละบิต
– ลบ	&& คำสั่ง and (ผลเป็นบูลีน)
* คูณ	|| คำสั่ง and (ผลเป็นบูลีน)
/ หาร	> มากกว่า
% เศษ	< น้อยกว่า
>> เลื่อนบิตไปทางขวา	>= มากกว่าหรือเท่ากับ
<< เลื่อนบิตไปทางซ้าย	<= น้อยกว่าหรือเท่ากับ
& คำสั่ง and แบบเปรียบเทียบทีละบิต	== เท่ากับ
| คำสั่ง or แบบเปรียบเทียบทีละบิต	!= ไม่เท่ากับ

เครื่องหมายใช้กับนิพจน์ 3 นิพจน์

แบบ E_1 ? E_2 : E_3

ในกรณี E_1 ไม่เท่ากับ 0 ผลจะเป็น E_2 มิเช่นนั้นจะเป็น E_3

< GREP >

โปรแกรมนี้ทำหน้าที่ค้นหาคำหรือข้อความในไฟล์ ลักษณะทั่วไปสามารถเขียนเป็น

GREP [แบบการค้นหา] คำหรือข้อความ [ชื่อไฟล์]

แบบการค้นหามีดังนี้คือ

-C แสดงจำนวนบรรทัดที่มีคำหรือข้อความนั้น

-D ค้นหาคำในไดเร็กทอรีปัจจุบันและในไดเร็กทอรีย่อยของไดเร็กทอรีปัจจุบัน

-I ถือตัวอักษรใหญ่เล็กเหมือนกัน

-L พิมพ์ชื่อไฟล์ที่มีคำหรือข้อความนั้น

-N พิมพ์หมายเลขบรรทัดที่มีคำหรือข้อความนั้น

-O แสดงผลเป็นแบบ UNIX

-R ตัวอักษรในไฟล์ถือเป็นตัวอักษรธรรม (ไม่ถือเป็นเท็กซ์)

-U กำหนดแบบการค้นหาโดยผู้ใช้ดังตัวอย่าง

ex) -UI ที่จอจะปรากฏช่องใส่ชื่อไฟล์ สมมุติว่าใส่ GREP1 เมื่อใช้ GREP1 ค้นหาคำแบบการค้นจะเป็นการค้นโดยถือเอาตัวอักษรใหญ่เล็กเหมือนกัน

-V พิมพ์บรรทัดที่ไม่มีคำหรือข้อความนั้น

-W ค้นหาคำที่ตรงกันจริงๆเช่น ให้คำ en ถ้าในไฟล์มีคำว่า end ถือว่าไม่ตรงกัน

-Z แสดงชื่อไฟล์ จำนวนบรรทัด และเลขบรรทัดที่มีคำหรือข้อความนั้น

คำหรือข้อความอาจเขียนให้อยู่ภายในเครื่องหมาย (") ได้ และในกรณีที่ใช้แบบการค้น -R ยังสามารถเขียนสัญลักษณ์หลัง -R เพื่อกำหนดแบบการค้นหาได้ดังต่อไปนี้ (ต้องเว้นห่างจาก R 1 ช่องไฟ)

^ หาคำหรือข้อความที่จุดเริ่มต้นของบรรทัด

$ หาคำหรือข้อความที่จุดท้ายบรรทัด

. สัญลักษณ์นี้ถือเป็นอักขระใดๆได้ทุกอักขระ

* ค้นหาคำที่อยู่หน้าเครื่องหมาย * ดังตัวอย่าง ถ้ากำหนด fo* จะค้นหาคำ f fo

+ ค้นหาคำที่อยู่หน้าเครื่องหมาย + ดังตัวอย่าง ถ้ากำหนด fo+ จะค้นหาคำ fo foo

[] แทนคำหรือข้อความเช่น [xyz],[a-z] (a ถึง z),[^a-z] (ไม่ใช่ a ถึง z]

/ คำที่จะให้ค้นหานั้นต้องเป็นตัวอักษรใหญ่หรือเล็กและอันเดอร์บาร์ ในกรณีที่ต้องการค้นหา . หรือ , ต้องใช้ / ใส่หน้า . หรือ ,

< TPC >

การคอมไพล์ภาษา Pascal นอกจากจะใช้ TURBO.EXE แล้ว ยังสามารถใช้โปรแกรม TPC.EXE ได้อีกด้วย คำสั่งนี้จะคอมไพล์โปรแกรมบน DOS และสร้างไฟล์.exe ซึ่งแบบการใช้งานคือ

TPC [แบบการคอมไพล์] ชื่อไฟล์ [แบบการคอมไพล์]

แบบ	ความหมาย
TPC /$A+ filename	{$A+}
หรือ TPC filename /$A-	
TPC filename /DIOCHECK;DEBUG	{$DEFINE IOCHECK}{$DEFINE DEBUG}
TPC filename /$M1600,0,655360	กำหนดขนาดขนาดแสตกและฮีพ
TPC filename /M	คอมไพล์โดยคำสั่ง MAKE
TPC filename /B	คอมไพล์โดยคำสั่ง BUILD
TPC filename /Fo:11	ค้นหา error ที่ตำแหน่ง o:11
TPC filename /L	OPtions/Linker/LinkBuffer memory
TPC /Q filename	ไม่แสดงรายละเอียดการคอมไพล์ออกจอ
TPC /TC:\tp filename	เก็บไฟล์ TPC.EXE และ TURBO.TPL ไว้ที่ ไดรฟ C ซับไดเร็กทอรี tp\bin
TPC /EC:\tp filename	กำหนดไดรฟ์เก็บไฟล์.EXE และไฟล์ .TPU
TPC /IC:\tp filename	กำหนดไดรฟ์ของไฟล์ที่ต้องการอ่าน (include)
TPC /UC:\tp filename	กำหนดไดร์ฟของไฟล์ยูนิต
TPC /OC:\tp filename	กำหนดไดร์ฟของไฟล์.OBJ
TPC/GS filename	/Options/LInker/Mapfile Segments
TPC/GP filename	/Options/LInker/Mapfile Publics
TPC/GD filename	/Options/LInker/Mapfile Detailed
TPC /V	สร้างไฟล์.exe ที่สามารถใช้ Turbo Debugger ตรวจหรือแก้ไขได้

< TPC.CFG >

การกำหนดวิธีการคอมไพล์โดยคำสั่ง TPC.EXE นอกจากจะใช้โดยตรงดังที่เห็นหน้าที่แล้ว อาจสร้างไฟล์เพื่อเก็บแบบการคอมไพล์ต่างๆลงไฟล์ TPC.CFG ดังตัวอย่าง

ex) /IC:\TP\INC (* เก็บไว้ในไฟล์ TPC.CFG *)

เมื่อเรียกใช้ TPC filename

จะมีความหมายเป็น TPC filename /IC:\TP\INC

< TPCONFIG.EXE >

โปรแกรมส่งแบบการเก็บข้อมูลระหว่างไฟล์.TP กับไฟล์ TPC.CFG โดยมีแบบการใช้งานดังนี้

ex) TPCONFIG TURBO.TP TPC.CFG

จากตัวอย่างข้างบน จะเป็นการส่งแบบการคอมไพล์จากไฟล์ TURBO.TP ให้กับไฟล์ TPC.CFG ในกรณีที่ไม่มีไฟล์ TPC.CFG จะสร้างไฟล์ TPC.CFG ขึ้นใหม่และสามารถส่งแบบคอมไพล์จากไฟล์ TPC.CFG ให้แก่ TURBO.TP ได้เช่นกัน

< UPGRADE >

โปรแกรมแปลงไฟล์ภาษา Pascal ซึ่งเขียนโดยใช้ Version 3.0 ซึ่งโปรแกรมนี้มีแบบการใช้งานดังนี้

UPGRADE [แบบการแก้ไข] ชื่อไฟล์

สำหรับแบบการแก้ไขมีดังนี้คือ

/3 ในกรณีที่ใช้คำสั่งของ Turbo3 จะใส่คำสั่ง uses turbo3 โดยอัตโนมัติ

/j ในกรณีที่ต้องการเขียนรายละเอียดลงไฟล์ชื่อเดียวกับโปรแกรม.JNL

/n ในกรณีที่ไม่ต้องการแสดงรายละเอียด

/O ในกรณีที่ต้องการบันทึกไฟล์ที่แก้ไขไปยัง PATH ที่ต้องการ ดังตัวอย่าง

ex) UPGRADE test c:\turbo5

/U เป็นการแบ่งไฟล์โปรแกรมเป็นยูนิตย่อยๆ โดยต้องเขียน .u ชื่อยูนิต ไว้ที่ต้นไฟล์

ภาคผนวก จ

< เฉลยแบบฝึกหัดท้ายบท >

ข้อ3.1

```
program summation;
var n,i,s:integer;
begin
    readln(n);
    s:=0;
    i:=0;
    while i<n do
       begin
          i:=i+1;
          s:=s+i
       end;
    writeln(s)
end.
```

ข้อ3.2

1) การเปลี่ยนแปลงของ i d s $\begin{bmatrix} i \\ d \\ s \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 2 \\ 5 \\ 4 \end{bmatrix} \rightarrow \begin{bmatrix} 3 \\ 7 \\ 9 \end{bmatrix} \rightarrow \begin{bmatrix} 4 \\ 9 \\ 16 \end{bmatrix} \rightarrow \begin{bmatrix} 5 \\ 11 \\ 25 \end{bmatrix}$

2) $s=i^2$

3) เนื่องจาก$(i+1)^2 = i^2+2i+1$ เมื่อ $d=2i+1$ แสดงว่า $s=i^2$ ทุกกรณี

ข้อ3.3

```
program divisor;
var i,j:integer;
begin
   readln(i);
   write('The factor of ',i,' are   1');
   for j:=2 to i do
      while i mod j = 0 do
         begin
            i:=i div j;
            write(' * ',j)
         end
end.
```

ข้อ3.4

```
program max_min;
var data,max1,max2,min1,min2:integer;
begin
   write('Input data ');
   readln(data);
   max1:=data; min1:=data; min2:=999; max2:=-999;
   repeat
      if data>max1 then begin
                           max2:=max1; max1:=data
                        end
      else if data>max2 then max2:=data
```

```
      else if (data>min1) and (data<min2) then  min2:=data
      else if data<min1 then begin
                                      min2:=min1; min1:=data
                                   end;
      readln(data);
   until data=-9999;
   writeln('maximum1 is ',max1);
   writeln('maximum2 is ',max2);
   writeln('minimum1 is ',min1);
   writeln('minimum2 is ',min2)
end.
```

ข้อ3.5

เอาท์พุทคือ 1 2 2 และ 2 1 2 ตามลำดับ

ข้อ 3.6

```
program birdturtle;
var number,legs,bird,turtle:integer;
begin
   write('Input total and legs ');
   read(number,legs);
   bird:=number; turtle:=0;
   while bird*2+turtle*4<legs do
      begin
         bird:=bird-1;
         turtle:=turtle+1
      end;
   if (bird*2+turtle*4=legs) and (bird>=0)
   then writeln('turtle= ',turtle,' bird= ',bird)
   else writeln('no answer')
end.
```

ข้อ4.1

```
program powerof2;
const n=100;
var a:array[1..n] of integer;
    i,j:integer;
begin
   a[1]:=1;
   for j:= 2 to n do
      a[j]:=0;
   for i:=1 to n*2 do
      for j:=n downto 1 do
         begin
            a[j]:=a[j]*2;
            if a[j]>9 then
                             begin
                               a[j+1]:=a[j+1]+a[j] div 10;
                               a[j]:=a[j] mod 10
                             end
         end;
   j:=n;
   while a[j]=0 do j:=j-1;
   for i:=j downto 1 do
      write(a[i])
end.
```

ข้อ4.2

```
program pascal_triangle;
const n=10;
var p,t:array[1..n] of integer;
    i,j:integer;
begin
  for i:=1 to n do
    begin
      for j:=2 to i-1 do
        p[j]:=t[j-1]+t[j];
      p[i]:=1;
      write(' ':3*(n-i+1));
      for j:=1 to i do
        begin
          write(p[j]:3,' ':3);
          t[j]:=p[j]
        end;
      writeln;
    end
end.
```

ข้อ4.3

```
program histogram;
const step=5;
var data:integer;
    i,j:integer;
    h:array[0..50] of integer;
begin
  for i:=0 to 50 do h[i]:=0;
  writeln('>>>>> input data <<<<<');
  readln(data);
  while data<>-999 do
    begin
      i:=data div step;
      h[i]:=h[i]+1;
      readln(data)
    end;
  for i:=25 to 40 do
    begin
      write(i*step,'..',i*step+step,h[i]:3,' person',' : ');
      for j:=1 to h[i] do write('*');
      writeln
    end
end.
```

ข้อ4.4

```
program linearequation;
const n=10;
var a:array[1..n,1..n] of real;
    y:array[1..n] of real;
    i,j,k,m:1..n;
begin
  write('input numbers of equations ');
  readln(m);
```

```
for i:=1 to m do
   for j:=1 to m do
      readln(a[i,j]);            { read matrix A }
for i:=1 to m do
   readln(y[i]);                 { read vector Y }
for i:=1 to m do
   begin
      for j:=1 to m do
         write(a[i,j]:10:2);
         writeln(y[i]:10:2)
   end;
{ solve AX = B }
for k:=1 to m do
   begin
      for j:=k+1 to m do
         a[k,j]:=a[k,j]/a[k,k];
         y[k]:=y[k]/a[k,k];
         for i:=1 to m do
            if i<>k then
               begin
                  for j:=k+1 to m do
                     a[i,j]:=a[i,i]-a[i,k]*a[k,j];
                     y[i]:=y[i]-a[i,k]*y[k]
               end
   end;
for i:=1 to m do writeln(y[i])
end.
```

ข้อ5.1

```
program primes;
const setsize=250;
      maxel=250;
      w=40;
var sieve,prime:array[0..w] of set of 0..maxel;
    an,bn:integer;
    j,k,s,t,count:integer;
    empty:boolean;
begin
   for t:=0 to w do
     begin
       sieve[t]:=[];
       for s:=0 to maxel do
          sieve[t]:=sieve[t]+[s];
          prime[t]:=[]
     end;
   sieve[0]:=sieve[0]-[0,1];
   bn:=0;
   an:=0;
   empty:=false;
   repeat
     while not(bn in sieve[an]) do
        bn:=bn+1;
     prime[an]:=prime[an]+[bn];
     j:=bn; k:=an;
```

```
          while k<=w do
             begin
                sieve[k]:=sieve[k]-[j];
                j:=j+bn;
                k:=k+an;
                while j>maxel do
                   begin
                      j:=j-setsize;
                      k:=k+1
                   end
             end;
          if sieve[an]=[] then
             begin
                bn:=0;
                empty:=true
             end;
          while (an<w) and empty do
             begin
                an:=an+1;
                empty:=sieve[an]=[]
             end
       until empty;
       { output }
       count:=0;
       for t:=0 to w do
          for s:=0 to maxel do
             if s in prime[t] then
                begin
                   count:=count+1;
                   write(t*setsize+s:5);
                   if count mod 10=0 then writeln
                end
    end.
```

ข้อ5.2

```
    program divide;
    const n=255;
    var p:set of 0..n;
        i,j:integer;
    begin
       p:=[2..n];
       for i:=2 to n do
          begin
             if i in p then
                for j:=32*i to n do
                   if (j mod 2=0) and
                      (j mod 3=0) and
                      (j mod 5<>0) and
                      (j mod 7<>0) and
                      (j mod 9<>0) then
                                          p:=p-[j]
          end;
       for i:=2 to n do
          if not (i in p) then write(i:4)
    end
```

ข้อ6.1

```
program marry;
type inform=record
          name,familyname   : string[10];
          date,month,year,age : integer;
          status            : string[10];
       end;
var person:array[1..2] of inform;
   i:integer;
begin
   for i:=1 to 2 do
      with person[i] do
         begin
           if i=1 then writeln('Please enter man''s data ')
                  else writeln('Please enter woman''s data ');
           write('name ');      readln(name);
           write('familyname ');readln(familyname);
           write('date ');      readln(date);
           write('month ');     readln(month);
           write('year ');      readln(year);
           write('age ');       readln(age);
           write('status ');    readln(status);
         end;
   if (person[1].status='s') and
      (person[2].status='s') and                  { s means single }
      (person[1].age>17)     and
      (person[2].age>15)     then
     begin
        person[1].status:='m';
        person[2].status:='m';                    { m means married }
        person[2].familyname:=person[1].familyname;
        writeln('Congrastulation !');
        write(person[1].name,'  ',person[1].familyname);
        write('      and ');
        write(person[2].name,'  ',person[2].familyname);
        write(' are married ');
     end
     else  writeln('Sorry they can''t marry')
end.
```

ข้อ6.2

```
program order;
const n=5;
type style = record
               name:string;
               bust,hip,waist:real;
            end;
var a:array[1..n] of style;
    dumy:style;
    i,L,k,s:integer;
    min:real;
begin
   writeln('bust hip waist name');
   for i:=1 to n do
      with a[i] do
         readln(bust,hip,waist,name);
```

```
for k:=1 to n-1 do
   begin
      min:=a[k].bust;
      s:=k;
      for L:=k+1 to n do
         if a[L].bust<=min then
            begin
               if a[L].bust=min then
                  if a[L].hip<=a[k].hip then
                     if a[L].hip<=a[k].hip then
                        if a[L].waist<a[k].waist then
                           begin
                              min:=a[L].bust;
                              s:=L
                           end
                        else
                     else
                        begin
                           min:=a[L].bust;
                           s:=L
                        end
                  else
               else
                  begin
                     min:=a[L].bust;
                     s:=L
                  end
            end;
      dumy:=a[k];
      a[k]:=a[s];
      a[s]:=dumy
   end;
   for i:=1 to n do
      with a[i] do
         writeln(name:12,bust:5:1,hip:5:1,waist:5:1)
end.
```

ข้อ6.3

```
program new_year_party;
type a=record
        name:string;
        money:integer;
        date:1..31;
     end;
var i,max,dumy:integer;
    man:array[1..50] of a;
    total:array[1..31] of integer;
begin
   writeln('input money date and name');
   i:=1;
   for i:=1 to 30 do
      total[i]:=0;
   max:=0;
```

```
    repeat
       with man[i] do
          begin
             read(money,date,name);
             total[date]:=total[date]+money;
             if total[date]>max then
                begin
                  max:=total[date];
                  dumy:=date
                end
          end;
    until man[i].money=0;
    writeln('party''s day is ',dumy,' total is ',max)
 end.
```

ข้อ7.1

```
 function combi(n,m:integer):integer;
 var i,s:integer;
 begin
    s:=1;
    for i:=1 to m do
       s:=s*(n-i+1) div i;
    combi:=s
 end;
```

ข้อ7.2

```
 program calculate;
 type fraction=record
                x:integer;
                y:integer
             end;
 var a,b,c:fraction;
     d:char;

   procedure compute(i,j:fraction;k:char;var L:fraction);
     var z:integer;
     begin
        case k of
           '+':begin
                  L.x:=i.x*j.y+j.x*i.y;
                  L.y:=i.y*j.y
               end;
           '-':begin
                  L.x:=i.x*j.y-j.x*i.y;
                  L.y:=i.y*j.y
               end;
           '*':begin
                  L.x:=i.x*j.x;
                  L.y:=i.y*j.y
               end;
           '/':begin
                  L.x:=i.x*j.y;
                  L.y:=i.y*j.x
               end;
```

```
   end;
   z:=L.x+1;
   repeat
      z:=z-1;
   until (L.x mod z=0) and (L.y mod z=0);
   L.x:=l.x div z;
   L.y:=L.y div z
  end;

begin
   writeln('input first fraction ');
   readln(a.x);
   write('/');
   readln(a.y);
   writeln('input operator');
   readln(d);
   writeln('input second fraction ');
   readln(b.x);
   write('/');
   readln(b.y);
   compute(a,b,d,c);
   writeln('answer is ',c.x,'/',c.y)
end.
```

ข้อ7.3

```
program  compute2;
uses  crt;
const  max = 125;
type digit = array[0..max] of integer;
var  b1,b2:boolean;  i,j,z,z1:integer;
     d1,d2:digit;  ch:char;

   procedure readinput(var b:boolean; var n:integer; var d:digit);
   var ch:char;
       i:integer;
   begin
      for i:=0 to z do d[i]:=0;  i:=1;
      b:=true; ch:=readkey; write(ch);
      if ch in ['-','0'..'9'] then
      begin
         if ch = '-' then
         begin
            b:= false; ch:=readkey;
            write(ch);
         end;
         while ch = '0' do
         begin
            ch:=readkey; write(ch);
         end;
         while ch in ['0'..'9'] do
         begin
            d[i]:=ord(ch)-ord('0');
            ch:=readkey; write(ch);
            i:=i+1
         end;
```

```
        n:=i-1; writeln;
        for i:=1 to n do
        begin
           d[z1-i]:=d[n-i+1];
           d[n-i+1]:=0;
        end;
     end else writeln('                ERROR          ');
  end;

  procedure check(var d:digit);
  var i,j:integer;
  begin
     j:=1;   i:=z;
     while (d[j]=0) and (j<>z) do j:=j+1;
     if (j=z) and (d[j]=0) then else
     repeat
      if d[i]>9 then
       begin
          d[i-1]:=d[i-1]+d[i] div 10;
          d[i]:=d[i] mod 10;
       end;
       if d[i]<0 then
       begin
          d[i-1]:=d[i-1]-1;
          d[i]:=10+d[i];
        end;
      i:=i-1;
     until i=j-1;
  end;

  procedure plus(d1,d2:digit; var d:digit);
  var i,j:integer;
  begin
     for j:=1 to z do d[j]:=d1[j]+d2[j];
     check(d);
  end;

  procedure minus(m,n:integer;d1,d2:digit;
                      var b:boolean;var d:digit);
  var i,j:integer;
  begin
    if m=n then
      begin
        i:=1;
        while (d1[i]=d2[i]) and (i<>z) do i:=i+1;
        if d1[i]>=d2[i] then
               for j:=1 to z do d[j]:=d1[j]-d2[j]
        else  if d1[i]<d[i] then
              begin
                for j:=1 to z do d[j]:=d2[j]-d1[j];
                  b:=false;
              end;
      end
    else if m<n then
         begin
           for j:=1 to z do d[j]:=d2[j]-d1[j];
           b:=false;
```

```
        end
   else for j:=1 to z do
          d[j]:=d1[j]-d2[j];
     check(d);
   end;

procedure multiplied(n1,n2:integer; d1,d2:digit; var d:digit);
var i,j:integer;
begin
  for i:=z1-n1 to z do
    for j:=z1-n2 to z do
      d[i+j-z]:=d1[i]*d2[j]+d[i+j-z];
      check(d);
end;

procedure divided(n1,n2:integer; d1,d2:digit;
                  var b:boolean; var d:digit);
var i,j,k,l:integer;
begin
   if n2 = 0 then write('   CAN NOT DIVIDE BY  ZERO   ')
   else  if  n1>=n2  then
     repeat
       k:=0;
       while d1[l]>=0 do
       begin
          for i:=z1-n1 to z-n1+n2  do
          begin
             j:=n1-n2+i;  d1[i]:=d1[i]-d2[j];
          end;
          check(d1); k:=k+1; l:=0;
          while d1[l]=0 do l:=l+1;
       end;
       if d1[l]<0 then
       begin
         for i:=z1-n1 to z-n1+n2  do
         begin
           j:=n1-n2+i;  d1[i]:=d1[i]+d2[j];
         end;
        check(d1);
       end;
       d[z-n1+n2]:=k-1;
       n1:=n1-1;
     until n1 < n2;
end;

  procedure think(op:char; b1,b2:boolean;
                  n1,n2:integer; d1,d2:digit);
  var i,j:integer; b:boolean; d:digit;
  begin
    for i:=0 to z do d[i]:=0;  b:=true;
    case op of
     '+' : if b1=b2 then begin plus(d1,d2,d); b:=b1 end
           else if b1 then minus(n1,n2,d1,d2,b,d)
           else minus(n2,n1,d2,d1,b,d);
     '-': if b1<>b2 then begin plus(d1,d2,d); b:=b1 end
           else if b1 then minus(n1,n2,d1,d2,b,d)
           else minus(n2,n1,d2,d1,b,d);
```

```
    '*' : if b1=b2 then multiplied(n1,n2,d1,d2,d)
          else  begin multiplied(n1,n2,d1,d2,d); b:=false  end;

    '/' : if b1=b2 then divided(n1,n2,d1,d2,b,d)
          else  begin divided(n1,n2,d1,d2,b,d); b:=false  end;

   end;
   write('RESULT        = ');
   if not b then write('-'); j:=1;
   while (d[j]=0) and (j<>z) do j:=j+1;
   for i:=j to z do write(d[i]); writeln
  end;

begin {main}
  repeat
    b1:=true; b2:=true;
    z:=125;  z1:=z+1;
    write('Input number and press return ');
    readinput(b1,i,d1);
    while not (ch in ['+','-','*','/']) do
    begin
      write('  +   -   *   /    : operator = ');
      readln(ch);
    end;
    readinput(b2,j,d2);
    think(ch,b1,b2,i,j,d1,d2);
    repeat
      write('ANOTHER   Y/N  ');
      readln(ch);
    until ch in ['y','Y','n','N'];
  until ch in ['n','N'];
end.
```

ข้อ8.1

```
program student;
uses crt;
type list_type = record
                    name     : string[16];
                    age      : byte;
                    grade    : byte;
                    address  : string[10]
                 end;

var
   list       : list_type;
   list_array : array [1..20] of list_type;
   list_file  : file              of list_type;
   select     : integer;

procedure list_input;
begin
   write('Please input name ');    readln(list.name);
   write('Please input age ');     readln(list.age);
   write('Please input grade ');   readln(list.grade);
   write('Please input address ');readln(list.address)
end;
```

```
procedure list_output(n:integer);
begin
    with list do
        begin
            write(n:4);
            write(' name ',name:16);
            write(' age  ',age:3);
            write(' class ',grade:2);
            writeln(' addres ',address:15)
        end
end;

procedure menu(var n:integer);
begin
    lowvideo;
    writeln;
    writeln(' 1 : Show all data');
    writeln(' 2 : Show one student');
    writeln(' 3 : Make new file');
    writeln(' 4 : Append');
    writeln(' 5 : Change data');
    writeln(' 0 : Exit');
    highvideo;
    writeln;
    writeln('Seclect any number');
    readln(n);
    writeln
end;

procedure print_all_member;
var i:integer;
begin
    reset(list_file);
    for i:=1 to filesize(list_file) do
        begin
            read(list_file,list);
            list_output(i)
        end
end;

procedure print_one_person;
var number:integer;
begin
    write('Which data do want to see ? ');
    readln(number);
    reset(list_file);
    if filesize(list_file) >= number then
        begin
            seek(list_file,number-1);
            read(list_file,list);
            list_output(number)
        end
end;
```

```
procedure make_newlist;
var counter,i:integer;
begin
   counter:=0;
   repeat
      list_input;
      counter:=counter+1;
      list_array[counter]:=list
   until (list.age=0) or (list.grade=0)
                      or (counter=20);
   if counter <> 20 then counter:=counter-1;
   rewrite(list_file);
   for i:=1 to counter do
      write(list_file,list_array[i])
end;
procedure add_one_person;
begin
   reset(list_file);
   list_input;
   seek(list_file,filesize(list_file));
   write(list_file,list)
end;
procedure correct_one_person;
var number:integer;
begin
   write('Which data do you want to change ? ');
   readln(number);
   if number <= filesize(list_file) then
      begin
         reset(list_file);
         seek(list_file,number-1);
         read(list_file,list);
         list_output(number);
         seek(list_file,number-1);
         list_input;
         write(list_file,list)
      end
end;
begin
   assign(list_file,'b:students.dat');
   repeat
      menu(select);
      case select of
         1:print_all_member;
         2:print_one_person;
         3:make_newlist;
         4:if filesize(list_file)<20 then add_one_person;
         5:correct_one_person
      end
   until select=0;
   close(list_file)
end.
```

ข้อ8.2

```
program telephone;
uses crt;
type directory=record
                    name:string[20];
                    num:string[10]
               end;
var tel    : file of directory;
    fna    : char;
    nt     : directory;
    check  : boolean;
    qp,nc,c: integer;
    rt,re  : char;

procedure readfile;
begin
   clrscr;
   lowvideo;
   writeln('     NAME            TEL');
   normvideo;
   c:=1;
   repeat
      read(tel,nt);
      nc:=filepos(tel);
      lowvideo;
      if nc>9 then write(nc,'.') else write('0',nc,'.');
      normvideo;
      write(nt.name);
      writeln(nt.num:34-length(nt.name));
      if nc=18*c then if not eof(tel) then
         begin
            writeln;
            writeln('please press return for showing next ');
            writeln('please press ''E'' for finish');
            read(re);
            if (re='e') or (re='E')
               then seek(tel,filesize(tel))
               else clrscr;
            c:=c+1;
         end;
   until eof(tel);
end;

procedure rewritefile;
begin
   gotoxy(42,5);
   lowvideo;
   writeln('------ change mode --------');
   gotoxy(42,6);
   highvideo;
   writeln('enter new tel-number');
   gotoxy(42,7);
   writeln('please input No',' 1-',nc);
   gotoxy(42,8);
   writeln('if you want to add press ',nc+1);
```

```
    repeat
       gotoxy(42,9);
       readln(qp);
    until qp<=nc+1;
    seek(tel,qp-1);
    gotoxy(42,10);
    if qp=nc+1 then writeln(' add new tel_number  --------')
                else writeln('change No',qp);
    gotoxy(42,12);
    writeln('NAME ?-----');
    gotoxy(42,13);
    readln(nt.name);
    gotoxy(42,14);
    writeln('TEL ?---');
    gotoxy(42,15);
    readln(nt.num);
    write(tel,nt);
end;

procedure writefile;
begin
    rewrite(tel);
    repeat
       writeln('please enter name');
       writeln('---NAME----------');
       readln(nt.name);
       writeln('please enter telephone number');
       readln(nt.num);
       write(tel,nt);
       writeln('finish ? y/n');
       readln(re);
    until (re='y') or (re='Y')
end;

procedure makefile;
begin
    clrscr;
    writeln('no file');
    writeln;
    writeln('Do you want to create new file ? y/n');
    readln(re);
    if (re='n') or (re='N')
       then begin end
       else begin
                writeln;
                assign(tel,fna+'.dta');
                {$I-}
                reset(tel);
                {$I+}
                check:=(IOresult=0);
                case check of
                   true:begin
                           writeln;
                           writeln('the file is present.');
                           writeln('Do you want to erase ? y/n');
                           readln(re);
```

```
                        if not (re='y') or (re='Y')
                        then begin end
                        else writefile;
                      end;
                 false:begin
                         writefile
                      end
            end
         end
end;

begin
   repeat
      clrscr;
      writeln('enter file''s name');
      readln(fna);
      assign(tel,fna+'.dta');
      {$I-}
      reset(tel);
      {$I+}
      check:=(IOresult=0);
         case check of
            true : begin
                      readfile;
                      writeln;
                      gotoxy(42,2);
                      writeln('Do you want to add any data ? y/n');
                      gotoxy(42,3);
                      readln(re);
                      if (re='n') or (re='N')
                      then begin end
                      else begin
                             writeln;
                             rewritefile;
                           end;
                   end;
            false :   makefile
         end;
      writeln;
      gotoxy(42,20);
      writeln('Do you want to see another ? y/n');
      gotoxy(42,21);
      readln(re);
   until (re='n') or (re='N');
   if re=upcase('y') then close(tel)
end.
```

ข้อ8.3

```
program Hotel_California;
uses   crt;
const MAXROOM = 10;
type   room=record
             num : integer;
             name,check_in,check_out: string;
          end;
      hotel = array[1..MAXROOM] of room;
```

```
var guest       : hotel;
    fivar       : file of hotel;
    filename    : string[12];
    check       : boolean;
    key         : char;
procedure writedata;
var i:integer;
begin
   clrscr;
   writeln('Welcome to hotel California !':50);
   for i := 1 to 2 do
    write('Number':10,'Name':10,'Check_in':10,'Check_out':10);
   writeln;
   for i := 1 to 80 do  write('-');
   for i := 1 to MAXROOM do
     with guest[i] do
        if (i mod 10) = 0
          then
             writeln(num:10,name:10,check_in:10,check_out:10)
          else
             write(num:10,name:10,check_in:10,check_out:10);
end;   { writedata }
procedure input;
var
  num,i  : integer;
  ch,key : char;
begin
   repeat
   repeat
     gotoxy(40,15);
     write('enter room number ');readln(num);
     gotoxy(40,16);
     write('enter guest''s name ');readln(guest[num].name);
     gotoxy(40,17);
     write('enter check in day ');readln(guest[num].check_in);
     gotoxy(40,18);
     write('enter check out day ');readln(guest[num].check_out);
     gotoxy(40,19);
     write('Do you want to input any more <Y/N> ');
     repeat ch:=readkey;until Upcase(ch) in ['Y','N'];
     writedata;
   until upcase(ch)='N';
     gotoxy(5,18);
     write('Press any key to add more data or Esc to Quit => ');
     key:=readkey; gotoxy(5,18);clreol;
   until key=£27;
end;


procedure checkfile;
begin
   clrscr;
   write('Please enter file''s name : '); readln(filename);
   filename := filename+'.dta';
   assign(fivar,filename);
```

```
   {$I-} reset(fivar); {$I+}
   check := (IOresult=0);
end;     { checkfile }

procedure showdata;
var i:integer;
begin
    checkfile;
    if check then
       begin
          read(fivar,guest);rewrite(fivar);
       end
          else begin
                 rewrite(fivar);
                 for i:=1 to 10 do
                 with guest[i] do
                 begin
                    num:=i;
                    name:='';
                    check_in:='';
                    check_out:=''
                 end;
               end;
    writedata;
    gotoxy(5,18);
    write('Press any key to add more data or Esc to Quit => ');
    key:=readkey; gotoxy(5,18);clreol;
    if key=£27 then exit;
    input; write(fivar,guest);
end; { showdata }

begin
   showdata;
   close(fivar)
end.
```

ข้อ9.1

```
program trigonometry;
var i:integer;
    j:real;
begin
   writeln(' degree          sin          cos                tan');
   writeln('------------------------------------------------------------');
   for i:=0 to 9 do
      begin
         j:=i*10*pi/180;
         writeln(i*10:5,sin(j):15:5,cos(j):15:5,sin(j)/cos(j):15:5)
      end;
   writeln('------------------------------------------------------------');
end.
```

ข้อ9.2

```
program power;
const n=10;
type digit=0..9;
```

```
var i,k,r:integer;
    d:array[1..n] of digit;
begin
   for  k:=1 to n do
     begin
        write('.');
        r:=0;
        for i:=1 to k-1 do
           begin
              r:=r*10+d[i];
              d[i]:=r div 2;
              r:=r-2*d[i];
              write(chr(d[i]+ord('0')))
           end;
        d[k]:=5;
        writeln('5')
     end
end.
```

ข้อ10.1

```
function gcd(m,n:integer):integer;
begin
   if n=0 then gcd:=m
          else gcd:=gcd(n,m mod n)
end;
```

ข้อ10.2

```
program checkqueen;
type    r          = array [1..8] of integer;
var     i,j,L,u    : integer;
        q          : boolean;
        a          : array [1..8] of boolean;
        b          : array [2..16] of boolean;
        c          : array [-7..7] of boolean;
        x,s,y      : r;
        z          : array [1..12,1..8] of integer;
function check(cs : r) : boolean;
var       ci,cj,ck   : integer;
          anr        : boolean;
begin
   ci:= 0;
   repeat
      ck:=0;
      ci:= ci+1;
      for cj:= 1 to 8 do
         if cs[cj] = z[ci,cj] then ck:= ck+1;
      if ck = 8 then anr:= false else anr:= true;
   until (ci = L) or (anr = false);
   check:= anr;
end;

procedure rotate (s : r; var ans : boolean);
var       m,n,p   : integer;
          t,v     : r;
```

```
begin
   p:= 1;
   for m:= 1 to 8 do v[m]:= s[m];
   repeat
     for m:= 1 to 8 do
     for n:= 1 to 8 do
        if v[n] =  m then t[m]:= 9-n;
     ans:= check(t);
     for m:= 1 to 8 do
        v[m]:= t[m];
     p:= p+1;
   until (p > 4) or (ans = false);
end;

procedure try(i : integer);
var       j,k  : integer;
begin
   for j:= 1 to 8 do
   if a[j] and b[i+j] and c[i-j] then
     begin
       x[i]:= j;
       a[j]:= false; b[i+j]:= false; c[i-j]:= false;
       if i<8 then try(i+1)
       else
       begin
         if L = 1 then
           begin
             for k:= 1 to 8 do z[1,k]:=x[k];
             L:= L+1;
           end
                     else
           begin
             L:= L+1;
             q:= true;
             rotate(x,q);
             if q = true then
               begin
                 for k:= 1 to 8 do
                   y[k]:= x[9-k];
                 rotate(y,q);
                 if q = true then
                   begin
                     for k:= 1 to 8 do
                       z[L-1,k]:=x[k];
                   end
                         else
                   L:= L-1;
               end
             else L:=L-1;
           end;
       end;
       a[j]:= true; b[i+j]:= true; c[i-j]:= true;
     end;
end;
```

```
begin
  L:= 1;
  for i:= 1 to 8 do a[i]:= true;
  for i:= 2 to 16 do b[i]:= true;
  for i:= -7 to 7 do c[i]:= true;
  try(1);
  for i:= 1 to L-1 do
     begin
        for u:= 1 to 8 do write(z[i,u]:3); writeln;
     end;
end.
```

ข้อ10.3

```
program index_file;
type list   = string[20];
     data   = ^person;
     person = record
                 name:list;
                 age:byte;
                 next:data
              end;
var fp :file of person;
    idx:file of list;
    head,now,last:data;
    n:person;
    i,count:integer;
begin
    assign(fp,'b:alldata');
    rewrite(fp);
    new(now);
    now^.name:='';
    now^.next:=nil;
    writeln('Input data from name and age ');
    readln(n.name);
    readln(n.age);
    while n.name<>'///' do
      begin
         write(fp,n);
         last:=now;
         while last^.name > n.name do
            last:=last^.next;
         new(head);
         head^:=last^;
         last^.name:=n.name;
         last^.next:=head;
         readln(n.name);
         readln(n.age);
      end;
    close(fp);
    assign(idx,'b:index.dta');          { Create index file }
    rewrite(idx);
    while now^.next <> nil do
       begin
         write(idx,now^.name);
         now:=now^.next
       end;
```

```
    reset(idx);
    while not eof(idx) do
      begin
        read(idx,n.name);
        writeln(n.name)
      end;
    close(idx)
  end.
```

ข้อ11.1

```
program star;
uses graph;
var iangle:real;
    i,j:integer;
{$i b:turtle.pas }
begin
    i:=detect;
    initgraph(i,j,'');
    iangle:=0;
    moveto(getmaxx div 2,getmaxy div 2);
    for i:=1 to 20 do
    begin
       turn(5);
       pen(100);
       turn(185);
       pen(100);
    end;
    readln;
    closegraph;
end.
```

ข้อ11.2

```
program rotate;
uses graph,crt;
type _2dpoint=record
                 x,y:integer
              end;
     _3dpoint=record
                 x,y,z:real
              end;
const   data : array[1..4,1..4] of _3dpoint =
(((x:-8;y:5;z:0),(x: 8;y:5;z: 0),(x: 8;y:0;z: 5),(x:-8;y:0;z: 5)),
 ((x:-8;y:5;z:0),(x: 8;y:5;z: 0),(x: 8;y:0;z:-5),(x:-8;y:0;z:-5)),
 ((x:-8;y:0;z:5),(x: 8;y:0;z: 5),(x: 8;y:0;z:-5),(x:-8;y:0;z:-5)),
 ((x:-8;y:0;z:5),(x:-8;y:0;z:-5),(x:-8;y:5;z: 0),(x:-8;y:0;z: 5)));
var  x_shift, y_shift, z_shift : real;
     x_sin, y_sin, z_sin, x_cos, y_cos, z_cos : real;
     i,j:integer;
     c:_3dpoint;
     d:array[1..4,1..4] of _2dpoint;
{$i b:graph4.pas }
begin
   i:=detect;
   initgraph(i,j,'');
   x_shift:=0.0; y_shift:=0.0; z_shift:=0.0;
   c.x:=0.0;     c.y:=0.0;      c.z:=10.0;
```

```
   repeat
      rotation(c);
      setcolor(1);
      for i := 1 to 4 do
         begin
            for j:=1 to 4 do
               begin
                  trans(data[i,j],data[i,j]);
                  purse(data[i,j],d[i,j]);
               end;
            drawpoly(4,d[i]);
         end;
      setcolor(0);
      for i := 1 to 4 do
         drawpoly(4,d[i]);
      c.x:=c.x+15;
      c.y:=c.y+15;
      c.z:=c.z+15;
   until keypressed or (c.z>360);
   readln;
   closegraph
end.
```

1.3

```
program box;
uses graph;
const pln_num = 6;
type _2dpoint=record
              x:integer; y:integer
           end;
      _3dpoint=record
               x:real;y:real;z:real
            end;
      datatype = array[1..6,1..10] of _3dpoint;
      order=array [1..pln_num] of integer;
const build:fillpatterntype=(0,0,0,0,0,0,0,0);
      pnt : array[1..6,1..10] of _3dpoint=
 (((x:-2;y:2;z:-2),(x:-2;y:-2;z:-2),(x:-2;y:-2;z:2),(x:-2;y:2;z:2),
   (x:-2;y:2;z:-2),(x:-2;y:1;z:-1),(x:-2;y:-1;z:-1),(x:-2;y:-1;z:1),
   (x:-2;y:1;z:1),(x:-2;y:1;z:-1)),
  ((x:-2;y:2;z:2),(x:-2;y:-2;z:2),(x:2;y:-2;z:2),(x:2;y:2;z:2),
   (x:-2;y:2;z:2),(x:-1;y:1;z:2),(x:-1;y:-1;z:2),(x:1;y:-1;z:2),
   (x:1;y:1;z:2),(x:-1;y:1;z:2)),
  ((x:2;y:2;z:2),(x:2;y:-2;z:2),(x:2;y:-2;z:-2),(x:2;y:2;z:-2),
   (x:2;y:2;z:2),(x:2;y:1;z:1),(x:2;y:-1;z:1),(x:2;y:-1;z:-1),
   (x:2;y:1;z:-1),(x:2;y:1;z:1)),
  ((x:2;y:2;z:-2),(x:2;y:-2;z:-2),(x:-2;y:-2;z:-2),(x:-2;y:2;z:-2),
   (x:2;y:2;z:-2),(x:1;y:1;z:-2),(x:1;y:-1;z:-2),(x:-1;y:-1;z:-2),
   (x:-1;y:1;z:-2),(x:1;y:1;z:-2)),
  ((x:-2;y:2;z:2),(x:2;y:2;z:2),(x:2;y:2;z:-2),(x:-2;y:2;z:-2),
   (x:-2;y:2;z:2),(x:-1;y:2;z:1),(x:1;y:2;z:1),(x:1;y:2;z:-1),
   (x:-1;y:2;z:-1),(x:-1;y:2;z:1)),
  ((x:-2;y:-2;z:2),(x:-2;y:-2;z:-2),(x:2;y:-2;z:-2),(x:2;y:-2;z:2),
   (x:-2;y:-2;z:2),(x:-1;y:-2;z:1),(x:-1;y:-2;z:-1),(x:1;y:-2;z:-1),
   (x:1;y:-2;z:1),(x:-1;y:-2;z:1)));
```

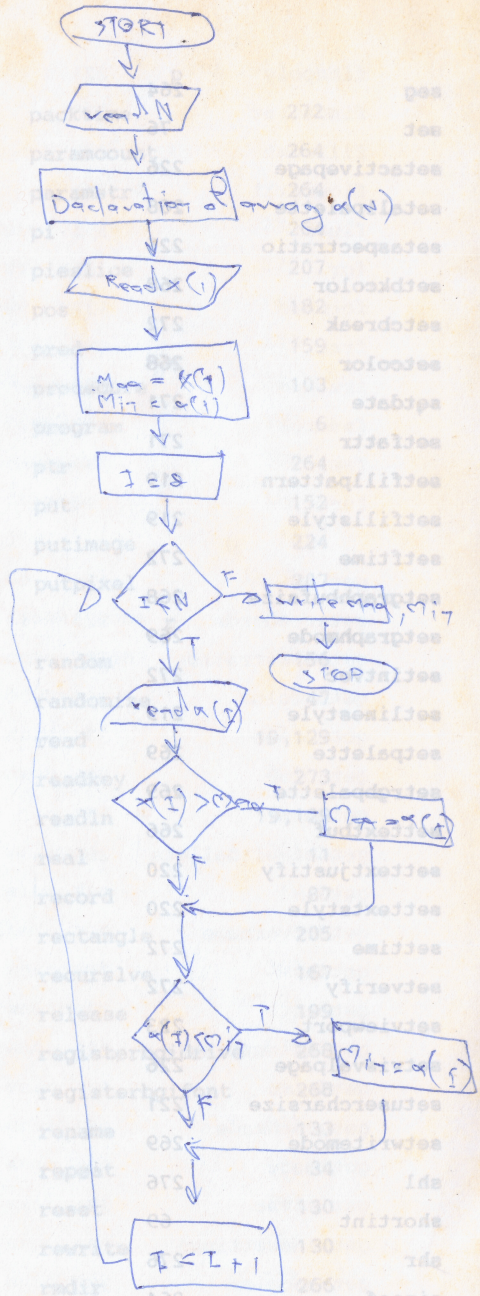
```
  var xs:array[1..pln_num,1..10] of _2dpoint;
      x_sin,y_sin,z_sin,x_cos,y_cos,z_cos,
      x_shift,y_shift,z_shift:real;
      prior,z_value:order;
      c,xx:_3dpoint;
      i,j,k,x,y,z,n,sum:integer;
      pntreal:array[1..pln_num,1..10] of _3dpoint;
  {$i b:graph4.pas }

procedure sort(z_value:order;var prior:order);
var flag:array[1..pln_num] of integer;
    i,j,min,num:integer;
begin
   for i:=1 to pln_num do
      flag[i]:=0;
   for i:=1 to pln_num do
      begin
         min:=10000;
         for j:=1 to pln_num do
            if (flag[j]=0) and (z_value[j]<min) then
               begin
                  min:=z_value[j];
                  num:=j
               end;
            prior[i]:=num;
            flag[num]:=1;
      end
end;
    procedure makedata;
       begin
          x_shift:=-5; y_shift:=5; z_shift:=20;
          for i:= 1 to pln_num do
             for j:=1 to 10 do
                trans(pnt[i,j],pntreal[i,j]);
       end;
  begin   { main}
    c.x:=30;c.y:=20;c.z:=10;
    rotation(c);    makedata;   (* call data *)
    i:=detect; initgraph(i,j,'');
    for i:=1 to pln_num do
       begin sum:=0;
             for j:=1 to 10 do
                begin
                   trans(pntreal[i,j],xx); purse(xx,xs[i,j]);
                   sum:=sum+round(xx.z)
                end;
             z_value[i]:=sum;
       end;
    sort(z_value,prior);
    for i:=1 to pln_num do
       begin
          n:=prior[i];
          setfillpattern(build,1);
          fillpoly(10,xs[n]);
       end;
    readln; closegraph;
  end.
```

ดัชนี

Mi Array

Mi Array



สำนักพิมพ์โอเดียนสโตร์

860-862 วังบูรพา กรุงเทพมหานคร 10200 โทร. 221-0742, 221-6567

218/10-12 สยามสแควร์ซอย 1 กรุงเทพมหานคร 10500 โทร. 251-4476, 254-8806 - 7